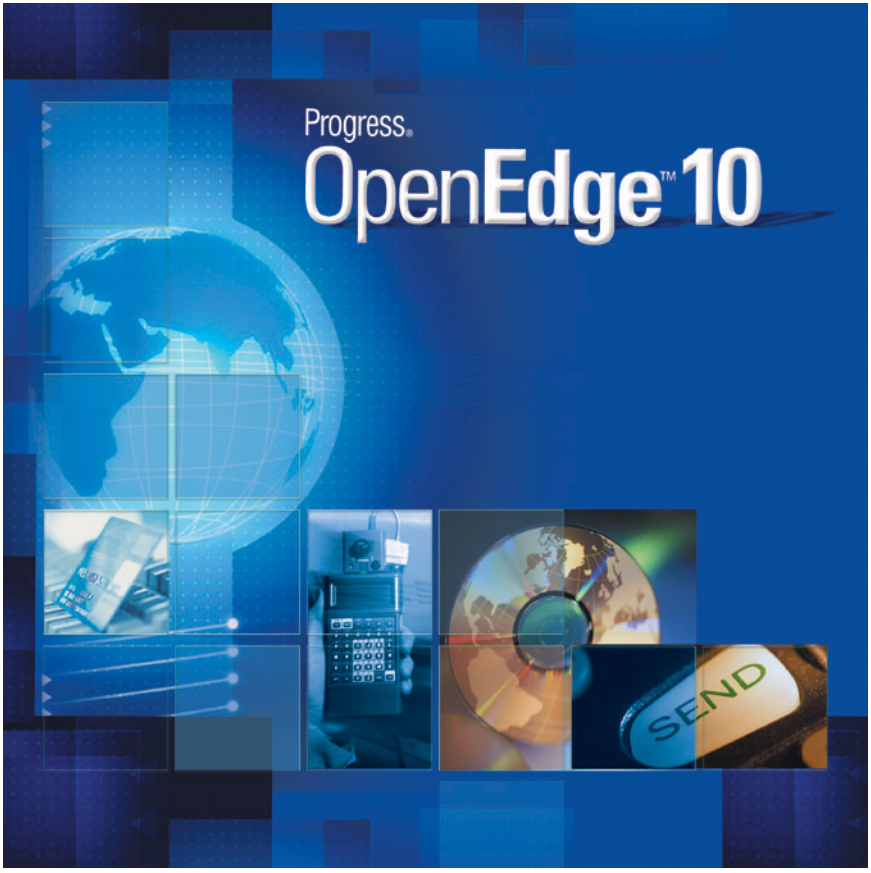


OpenEdge Development: Progress 4GL Reference

© 2005 Progress Software Corporation. All rights reserved.

Progress® software products are copyrighted and all rights are reserved by Progress Software Corporation. This manual is also copyrighted and all rights are reserved. This manual may not, in whole or in part, be copied, photocopied, translated, or reduced to any electronic medium or machine-readable form without prior consent, in writing, from Progress Software Corporation. The information in this manual is subject to change without notice, and Progress Software Corporation assumes no responsibility for any errors that may appear in this document. The references in this manual to specific platforms supported are subject to change.

A (and design), Allegrix, Allegrix (and design), Business Empowerment, DataDirect (and design) DataDrirect Connect, DataDirect OLE DB, DirectALert, EdgeXtend, Empowerment Center, eXcelon, IntelliStream, O (and design), ObjectStore, OpenEdge, PeerDirect, P.I.P., POSSENET, Powered by Progress, Progress, Progress Dynamics, Progress Empowerment Center, Progress Empowerment Program, Progress Fast Track, Progress OpenEdge, Partners in Progress, Partners en Progress, Persistence, Persistence (and design), ProCare, Progress en Partners, Progress in Progress, Progress Profiles, Progress Results, Progress Software Developers Network, ProtoSpeed, ProVision, SequeLink, SmartBeans, SpeedScript, Stylus Studio, Technical Empowerment, WebSpeed and Your Software, Our Technology–Experience the Connection are registered trademarks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and/or other countries. AccelEvent, A Data Center of Your Very Own, AppsAlive, AppServer, ASPen, ASP-in-a-Box, BusinessEdge, Cache-Forward, DataDirect, DataDirect Connect64, DataDirect Technologies, Data Direct XQuery, DataXtend, Fathom, Future Proof, ObjectCache, ObjectStore Event Engine, ObjectStore Inspector, ObjectStore Performance Expert, ObjectStore Trading Accelerator, POSSE, ProDataSet, Progress Business Empowerment, Progress DataXtend, Progress for Partners, Progress ObjectStore, PSE Pro, PS Select, SectorAlliance, SmartBrowser, SmartComponent, SmartDataBrowser, SmartDataObjects, SmartDataView, SmartDialog, SmartFolder, SmartFrame, SmartObjects, SmartPanel, SmartQuery, SmartViewer, SmartWindow, WebClient, and Who Makes Progress are trademarks or service marks of Progress Software Corporation or one of its subsidiaries or affiliates in the U.S. and other countries. SonicMQ is a registered trademark of Sonic Software Corporation in the U.S. and other countries. Vermont Views is a registered trademark of Vermont Creative Software in the U.S. and other countries. IBM is a registered trademark of IBM Corporation. JMX and JMX-based marks and Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Any other trademarks or service marks contained herein are the property of their respective owners.

OpenEdge includes Imaging Technology copyrighted by Snowbound Software 1993-2003. www.snowbound.com.

OpenEdge includes software developed by the Apache Software Foundation (http://www.apache.org/). Copyright © 1999 The Apache Software Foundation. All rights reserved (Xerces C++ Parser (XML) and Xerces2 Java Parser (XML)); Copyright © 1999-2002 The Apache Software Foundation. All rights reserved (Xerces Parser (XML)); and Copyright © 2000-2003. The Apache Software Foundation. All rights reserved (Ant). The names “Apache,” “Xerces,” “ANT,” and “Apache Software Foundation” must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called “Apache”, nor may “Apache” appear in their name, without prior written permission of the Apache Software Foundation. For written permission, please contact apache@apache.org. Software distributed on an “AS IS” basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License agreement that accompanies the product.

OpenEdge includes software developed by Vermont Creative Software. Copyright © 1988-1991 by Vermont Creative Software.

OpenEdge includes software developed by IBM and others. Copyright © 1999, International Business Machines Corporation and others. All rights reserved.

OpenEdge includes code licensed from RSA Security, Inc. Some portions licensed from IBM are available at http://oss.software.ibm.com/icu4j/.

OpenEdge includes the UnixWare platform of Perl Runtime authored by Kiem-Phong Vo and David Korn. Copyright © 1991, 1996 by AT&T Labs. Permission to use, copy, modify, and distribute this software for any purpose without fee is hereby granted, provided that this entire notice is included in all copies of any software which is or includes a copy or modification of this software and in all copies of the supporting documentation for such software. THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY. IN PARTICULAR, NEITHER THE AUTHORS NOR AT&T LABS MAKE ANY REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.

OpenEdge includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm. Copyright ©1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.

OpenEdge includes software developed by the World Wide Web Consortium. Copyright © 1994-2002 World Wide Web Consortium, (Massachusetts Institute of Technology, European Research Consortium for Informatics and Mathematics, Keio University). All rights reserved. This work is distributed under the W3C® Software License [http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231] in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

OpenEdge includes code licensed from Mort Bay Consulting Pty. Ltd. The Jetty Package is Copyright © 1998 Mort Bay Consulting Pty. Ltd. (Australia) and others.

OpenEdge includes the JMX Technology from Sun Microsystems, Inc.

OpenEdge includes software developed by the ModelObjects Group (http://www.modelobjects.com). Copyright © 2000-2001 ModelObjects Group. All rights reserved. The name "ModelObjects" must not be used to endorse or promote products derived from this software without prior written permission. Products derived from this software may not be called "ModelObjects", nor may "ModelObjects" appear in their name, without prior written permission. For written permission, please contact djacobs@modelobjects.com.

OpenEdge includes files that are subject to the Netscape Public License Version 1.1 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at http://www.mozilla.org/NPL/. Software distributed under the License is distributed on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the License for the specific language governing rights and limitations under the License. The Original Code is Mozilla Communicator client code, released March 31, 1998. The Initial Developer of the Original Code is Netscape Communications Corporation. Portions created by Netscape are Copyright © 1998-1999 Netscape Communications Corporation. All Rights Reserved.

December 2005

Product Code: 4496
R10.1A

Contents

Tables

Procedures

Preface

This Preface contains the following sections:

Purpose

This book defines the Progress® 4GL. It covers all 4GL statements, functions, phrases, operators, preprocessor directives, special symbols, widgets, handles, attributes, methods, events, and classes.

Audience

This book is intended for programmers who develop applications using Progress and for anyone who needs to read and understand Progress 4GL code.

Organization

This book consists of the following sections:

- A dictionary of Progress statements, functions, phrases, operators, preprocessors, and special symbols beginning with the letters A–Z.
- A dictionary of Progress widgets.
- A dictionary of Progress handles.
- A dictionary of Progress attributes and methods.
- Tables of Progress events.
- A dictionary of Progress classes.
- An index of Progress 4GL keywords.

Using this manual

Each Progress language element reference description includes some subset of the following information:

- Platform-restriction notations.
- A purpose or description of the language element.
- Block properties for all block statements.
- Data-movement diagrams for all data-handling statements.
- The syntax for the language element.
- The options and arguments you can use with the language element.
- One or more examples that illustrate the use of the language element.
- Notes that highlight special cases or provide hints on using the language element.
- A See Also section that lists related language elements.

Platform-restriction notes

Some language elements and features of the Progress 4GL do not apply to all software platforms—operating systems, user interfaces, and database management systems—that OpenEdge® supports. The documentation tries to note each such platform restriction with the language element title. Some language elements apply to SpeedScript programming and some do not; the documentation indicates which language elements do not apply with a note in the language element description.

You can consider a language element as supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in the language element description.

The platform restriction notes that appear in the documentation include the following:

- **AppServer™ only**

  The element or feature applies only to the OpenEdge AppServer.

- **Character interfaces only**

  The element or feature applies only to the character interfaces that OpenEdge supports.

- **Graphical interfaces only**

  The element or feature applies only to the graphical interfaces that OpenEdge supports.

- **NT and UNIX only**

  The element or feature applies only to the Windows and UNIX versions that OpenEdge supports.

- **ORACLE only**

  The element or feature applies only to the ORACLE versions that OpenEdge supports.

- **UNIX only**

  The element or feature applies only to the UNIX versions that OpenEdge supports.

- **Windows only**

  The element or feature applies only to the Windows versions that OpenEdge supports.

- **Windows only; Graphical interfaces only**

  The element or feature applies only to the graphical interfaces of the Windows and versions that OpenEdge supports.

For a complete list of the software platforms that OpenEdge supports, see *OpenEdge Getting Started: Installation and Configuration*.

Typographical conventions

This manual uses the following typographical conventions:

Convention	Description
Bold	Bold typeface indicates commands or characters the user types, provides emphasis, or the names of user interface elements.
Italic	Italic typeface indicates the title of a document, or signifies new terms.
SMALL, BOLD CAPITAL LETTERS	Small, bold capital letters indicate OpenEdge® key functions and generic keyboard keys; for example, **GET** and **CTRL**.
KEY1+KEY2	A plus sign between key names indicates a **simultaneous** key sequence: you press and hold down the first key while pressing the second key. For example, **CTRL+X**.
KEY1 KEY2	A space between key names indicates a **sequential** key sequence: you press and release the first key, then press another key. For example, **ESCAPE H**.
Syntax:	
`Fixed width`	A fixed-width font is used in syntax statements, code examples, system output, and filenames.
`Fixed-width italics`	Fixed-width italics indicate variables in syntax statements.
`Fixed-width bold`	Fixed-width bold indicates variables with special emphasis.
`UPPERCASE fixed width`	Uppercase words are Progress® 4GL language keywords. Although these are always shown in uppercase, you can type them in either uppercase or lowercase in a procedure.
	This icon (three arrows) introduces a multi-step procedure.
	This icon (one arrow) introduces a single-step procedure.
Period (.) or colon (:)	All statements except `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` end with a period. `DO`, `FOR`, `FUNCTION`, `PROCEDURE`, and `REPEAT` statements can end with either a period or a colon.

Convention	Description
[ ]	Large brackets indicate the items within them are optional.
[ ]	Small brackets are part of the Progress 4GL language.
{ }	Large braces indicate the items within them are required. They are used to simplify complex syntax diagrams.
{ }	Small braces are part of the Progress 4GL language. For example, a called external procedure must use braces when referencing arguments passed by a calling procedure.
|	A vertical bar indicates a choice.
...	Ellipses indicate repetition: you can choose one or more of the preceding items.

Examples of syntax descriptions

In this example, ACCUM is a keyword, and *aggregate* and *expression* are variables:

Syntax

```
ACCUM aggregate expression
```

FOR is one of the statements that can end with either a period or a colon, as in this example:

```
FOR EACH Customer:
  DISPLAY Name.
END.
```

In this example, STREAM *stream*, UNLESS-HIDDEN, and NO-ERROR are optional:

Syntax

```
DISPLAY [ STREAM stream ] [ UNLESS-HIDDEN ] [ NO-ERROR ]
```

In this example, the outer (small) brackets are part of the language, and the inner (large) brackets denote an optional item:

Syntax

```
INITIAL [ constant [ , constant ] ]
```

A called external procedure must use braces when referencing compile-time arguments passed by a calling procedure, as shown in this example:

Syntax

```
{ &argument-name }
```

In this example, EACH, FIRST, and LAST are optional, but you can choose only one of them:

Syntax

```
PRESELECT [ EACH | FIRST | LAST ] record-phrase
```

In this example, you must include two expressions, and optionally you can include more. Multiple expressions are separated by commas:

Syntax

```
MAXIMUM ( expression , expression [ , expression ] ... )
```

In this example, you must specify MESSAGE and at least one *expression* or SKIP [(*n*)], and any number of additional *expression* or SKIP [(*n*)] is allowed:

Syntax

```
MESSAGE { expression | SKIP [ ( n ) ] } ...
```

In this example, you must specify {*include-file*, then optionally any number of *argument* or *&argument-name = "argument-value"*, and then terminate with }:

Syntax

```
{ include-file
    [ argument | &argument-name = "argument-value" ] ... }
```

Long syntax descriptions split across lines

Some syntax descriptions are too long to fit on one line. When syntax descriptions are split across multiple lines, groups of optional and groups of required items are kept together in the required order.

In this example, WITH is followed by six optional items:

Syntax

```
WITH [ ACCUM max-length ] [ expression DOWN ]
  [ CENTERED ] [ n COLUMNS ] [ SIDE-LABELS ]
  [ STREAM-IO ]
```

Complex syntax descriptions with both required and optional elements

Some syntax descriptions are too complex to distinguish required and optional elements by bracketing only the optional elements. For such syntax, the descriptions include both braces (for required elements) and brackets (for optional elements).

In this example, ASSIGN requires either one or more *field* entries or one *record*. Options available with *field* or *record* are grouped with braces and brackets:

Syntax

```
ASSIGN   { [ FRAME frame ] { field [ = expression ] }
           [ WHEN expression ] } ...
       | { record [ EXCEPT field ... ] }
```

Example procedures

This manual provides numerous example procedures that illustrate syntax and concepts. You can access the example files and details for installing the examples from the following locations:

- The Documentation and Samples CD that you received with your product.
- The Progress Documentation Web site:

```
http://www.progress.com/products/documentation
```

OpenEdge messages

OpenEdge displays several types of messages to inform you of routine and unusual occurrences:

- **Execution messages** inform you of errors encountered while OpenEdge is running a procedure; for example, if OpenEdge cannot find a record with a specified index field value.
- **Compile messages** inform you of errors found while OpenEdge is reading and analyzing a procedure before running it; for example, if a procedure references a table name that is not defined in the database.
- **Startup messages** inform you of unusual conditions detected while OpenEdge is getting ready to execute; for example, if you entered an invalid startup parameter.

After displaying a message, OpenEdge proceeds in one of several ways:

- Continues execution, subject to the error-processing actions that you specify or that are assumed as part of the procedure. This is the most common action taken after execution messages.
- Returns to the Progress Procedure Editor, so you can correct an error in a procedure. This is the usual action taken after compiler messages.
- Halts processing of a procedure and returns immediately to the Progress Procedure Editor. This does not happen often.
- Terminates the current session.

OpenEdge messages end with a message number in parentheses. In this example, the message number is 200:

```
** Unknown table name table. (200)
```

If you encounter an error that terminates OpenEdge, note the message number before restarting.

Obtaining more information about OpenEdge messages

In Windows platforms, use OpenEdge online help to obtain more information about OpenEdge messages. Many OpenEdge tools include the following Help menu options to provide information about messages:

- Choose **Help→Recent Messages** to display detailed descriptions of the most recent OpenEdge message and all other messages returned in the current session.
- Choose **Help→Messages** and then type the message number to display a description of a specific OpenEdge message.
- In the Progress Procedure Editor, press the **HELP** key or **F1**.

On UNIX platforms, use the Progress `pro` command to start a single-user mode character OpenEdge client session and view a brief description of a message by providing its number.

To use the pro command to obtain a message description by message number:

1. Start the Progress Procedure Editor:

```
install-dir/dlc/bin/pro
```

2. Press **F3** to access the menu bar, then choose **Help→Messages**.
3. Type the message number and press **ENTER**. Details about that message number appear.
4. Press **F4** to close the message, press **F3** to access the Progress Procedure Editor menu, and choose **File→Exit**.

4GL Reference

This section contains reference entries that describe the Progress® language. They begin with descriptions of the language punctuation and special characters. The remaining entries contain descriptions of the Progress statements, functions, phrases, preprocessor directives, and miscellaneous other language elements.

You can consider a language element as supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in the language element description.

: Punctuation

The colon (:) symbol ends block labels and block header statements like DO, FOR, and REPEAT. It serves as a separator between a widget reference and an attribute or method, and between a character string literal and its attributes. It also follows the EDITING keyword in an EDITING phrase and is a device delimiter in Windows.

; Special character

The semicolon (;), when combined with a second character in the Procedure Editor, provides alternative representations of special characters as follows:

Special Character	@	[	]	^	'	{	|	}	~
Alternative Representation	;&	;<	;>	;*	;'	;(	;%	;)	;?

To suppress the semicolon's interpretation as a special character, precede it with a tilde (~). For example, to enter the string **;<** in the Procedure Editor and not have Progress interpret it as an open bracket, type **~;<.**

Additionally, if an ASCII character is mapped to an extended alphabetical character by an IN statement in the PROTERMCAP file, you can enter the extended character in the Procedure Editor by preceding the ASCII character with a semicolon. For example, if [is mapped to Ä, Progress interprets the **;[** sequence as Ä.

. Punctuation

The period (.) symbol ends all statements, including block header statements. It is also a directory path or file suffix separator in most platforms. The DO, FOR, and REPEAT statements can end with a period or a colon.

; Punctuation

In Progress Version 6.0 or later, the ANSI SQL (-Q) startup parameter allows you to redefine the semicolon as a terminator. This startup parameter enforces strict ANSI SQL conformance and allows you to terminate SQL statements with a semicolon. The ANSI SQL (-Q) parameter allows Progress to run standard SQL statements and scripts built with other products.

The ANSI SQL (-Q) parameter disables the use of the semicolon within UNIX escapes.

```
UNIX SMBL=foo; export SMBL
```

As a general rule, use the period (.) as a terminator for Progress statements even when you specify the ANSI SQL (-Q) parameter for an OpenEdge® session.

, Punctuation

The comma (,) symbol separates multiple file specifications (used in FOR statements, FOR phrases of DO and REPEAT statements, and PRESELECT phrases), branching statements (used in UNDO statements and phrases), and multiple arguments of a function.

? Special character

The question mark is a special character that represents the Unknown value. Progress treats a quoted question mark ("?") in a procedure or an input field as a question mark character. It treats an unquoted question mark (?) in a procedure or an input field as an unknown value.

Table 1 indicates the results when using the Unknown value (?) in a comparison expression (EQ, GE, GT, LE, LT, NE). These results are true for both character and integer variables.

Table 1: Using the Unknown value (?) in comparison operations

Comparison operator	One argument is ?	Both arguments are ?
EQ or =	F	T
GE or >=	?	T
GT or >	?	F
LE or <=	?	T
LT or <	?	F
NE or <>	T	F

Note: WebSpeed® treats an unquoted question mark (?) in an HTML input field as a character.

Additional points about the Unknown value (?) are:

- Any number of Unknown value (?) records can be in a unique index. This is useful in cases where you want to defer choosing key values for a unique index.
- If you define a field as mandatory in the Dictionary, that field cannot contain the Unknown value (?) when OpenEdge writes the record to the database.
- For sorting and indexing purposes, the Unknown value (?) sorts high.

- The question mark (?) character in the first position of a field equals the Unknown value (?), not a question mark.
- When using the Unknown value (?) in a comparison expression for SQL, the result is unknown.
- When using the Unknown value (?) in an expression, the result of that expression is usually unknown. For example, when you concatenate first, middle, and last names, and the middle name is ?, then the result is ?.

For information on how the Unknown value (?) works with logical data types, comparison operators, and conditional statements, see the following reference entries: EQ or = operator, GE or >= operator, GT or > operator, IF...THEN...ELSE statement, LE or < = operator, LT or < operator, NE or <> operator.

\ Special character

The backslash (\) is an escape character on UNIX platforms only. It is a directory path separator in Windows platforms only.

~ Special character

The tilde (~) is an escape character that causes Progress to read the following character literally. A tilde followed by three octal digits represents a single character. Use it as a lead-in to enter the special characters shown in Table 2. In a procedure, a tilde followed by something other than the items in Table 2 is ignored. For example, "~abc" is treated as "abc". (This may not work as expected when passing parameters to an include file.) The items in Table 2 are case sensitive.

Table 2: Entering special characters in the Procedure Editor *(1 of 2)*

Sequence	Interpreted as	Comment
~"	"	Use within quoted strings as an alternative to two quotes ("").
~'	'	Use within quoted strings as an alternative to two apostrophes ('').
~~	~	–
~\	\	–

Table 2: Entering special characters in the Procedure Editor *(2 of 2)*

Sequence	Interpreted as	Comment
~{	{	–
~*nnn*	A single character	Where *nnn* is an octal value between 000 and 377. All three digits are required.
~t	Tab character	Octal 011
~r	Carriage return	Octal 015
~n	New line / Line feed	Octal 012
~E	Escape	Octal 033
~b	Backspace	Octal 010
~f	Form feed	Octal 014

” Special character

The double quote (”) encloses character constants or strings. To use quotes within a quoted character string, you must use two double quotes (””), which compile to a single double quote (”), or you must put a tilde (~) in front of any quotes within the quoted character string. (This does not work when passing parameters to an include file.)

See also “ ” Character-string literal.

’ Special character

The function of the single quote (’) is the same as the double quote. But, if you use single and double quotes in a statement, the compiler checks the outermost quotes first, giving them precedence over the innermost quotes. For example, DISPLAY ’"test"’ returns as "test". (Progress reads the double quotes literally.) And DISPLAY "’test2’" returns as ’test2’.

See also “ ” Character-string literal.

/ Special character

The slash (/) symbol is a directory path separator (UNIX). It is also used for date fields (99/99/99).

See also “ ” Character-string literal.

( ) Expression precedence

Parentheses raise expression precedence. Also, some functions require you to enclose arguments in parentheses.

See also / Division operator.

[ ] Array reference

Square brackets ([ ]) enclose array subscripts ([1], [2], etc.) or ranges (such as, [1 FOR 4]). In a range, you can use a variable for the first element, but the second element must be a constant. The specification [1 FOR 4] causes Progress to start with the first array element and to work with that and the next three elements. Square brackets are also used when specifying initial values for an array. For example, if you define an array variable of extent 3, you might specify initial values as INITIAL [0, 1, 2].

= Special character

See “EQ or = operator”, “= Assignment operator”.

< Special character

See “LT or < operator”.

< = Special character

See “LE or < = operator”.

< > Special character

See “NE or <> operator”.

> Special character

See “GT or > operator”.

> = Special character

See “GE or >= operator”.

“ ” Character-string literal

Specifies a literal character-string value.

Syntax

```
“characters“ [ : [ R | L | C | T ] [ U ] [ max-length ] ]
```

characters

The literal contents of the character string.

R | L | C | T

Specifies the justification of the string within its maximum length: right, left, centered, or trimmed, respectively. The default justification depends on how the string is used. If the string is displayed with side labels, the default is right justification. If column labels are used, the defaults are left justification for character fields and right justification for numeric fields. Strings used in expressions are trimmed by default.

- R means right justified and padded on the left with spaces. “Hello”:R10 = “ Hello”.
- L means left justified and padded on the right with spaces. “Hello”:L10 = “Hello “.
- C means centered within the string and padded on both the right and left as needed. “Hello”:C10 = “ Hello “.
- T means trimmed of leading and trailing blanks (although storage space and screen space is still allocated for the maximum number of characters). “ Hello:T10 = “Hello” (but screen and storage space is still reserved for 10 characters).

U

Specifies that the string is untranslatable. This means that the string will not be processed by the OpenEdge Translation Manager. If you do not specify U, then the string is assumed to be translatable.

max-length

The number of characters reserved for the string contents in the text segment. The default is the length of the string itself. You might want to specify a longer length if you expect a translation of the string to be longer. The longest length you can specify is 5120 characters.

Note

If you include the colon (:) after the quoted string, you must supply at least one option. Otherwise, Progress treats the colon as a statement separator.

{ } Argument reference

References the value of an argument that a procedure passes to a called external procedure file or to an include file.

Progress converts each argument to a character format. This conversion removes the surrounding double-quotes if the parameter was specified as a character string constant in the RUN statement or include file reference.

When one procedure is called from another and arguments are used, Progress recompiles the called procedure, substituting the arguments that the calling procedure passes, and then runs the called procedure.

Syntax

```
{ { n | &argument-name } }
```

Enter the braces ({ }) as shown; they do not represent syntax notation in this description.

n

The number of the argument being referred to. If *n* = 0, Progress substitutes the name of the current procedure (the name you used when you called it, not the full pathname) as the argument. If *n* = *, Progress substitutes all arguments that the calling procedure passes (but not the name {0}). If you refer to the *n*th parameter and the calling procedure does not supply it, {*n*} is ignored.

&*argument-name*

The name of the argument being referred to. If you refer to an *argument-name* and the calling procedure does not supply it, Progress ignores {&*argument-name*}.

If *argument-name* is an asterisk (*), Progress substitutes all arguments that the calling procedure passes. It also adds quotation marks to each parameter, so you can pass the named argument list through multiple levels of include files.

Note: It is invalid to pass both numbered and named arguments within a single pair of braces. Although this will not cause a compile-time or run-time error, the arguments will not be passed correctly.

Examples

The procedure r-arg.p runs procedure r-arg2.p, passing the arguments customer and name to r-arg2.p. Progress substitutes these arguments for {1} and {2} in the r-arg2.p procedure:

r-arg.p

```
RUN r-arg2.p "customer" "name"
```

r-arg2.p

```
FOR EACH {1}:
  DISPLAY {2}.
END.
```

The r-inc.p procedure defines the variables txt and num, and assigns the values Progress VERSION and 7 to them. The r-inc.p procedure includes the r-inc.ifile and passes the &int and &str arguments to the include file. Because the parameters are named, their order is unimportant. The called procedure can find each argument, regardless of placement. The r-inc.i include file displays a message that consists of the passed arguments. The asterisk argument displays all the parameters as they are listed in the r-inc.p procedure:

r-inc.p

```
DEFINE VARIABLE txt AS CHARACTER.
DEFINE VARIABLE num AS INTEGER.

txt = "Progress VERSION".
num = 7.

{r-inc.i &int=num &str=txt}
```

r-inc.i

```
MESSAGE {&str}     /* the &str named argument */
        {&int}.    /* the &int named argument */

MESSAGE "An asterisk displays all the arguments:"
        {*}        /* all the arguments passed by the calling procedure */
```

Notes

- If you pass { } arguments using the RUN statement, you cannot precompile the called procedure. When Progress compiles a procedure, it must have all the values the procedure needs. So, if you pass arguments to a procedure you are calling with the RUN statement, Progress evaluates those arguments when the calling procedure is run, not when it is compiled.
- You can use the name of an include file as an argument to another include file. For example, a reference to {{1}} in an included procedure causes Progress to include the statements from the file with the name that passed as the first argument.
- Use DEFINE PARAMETER to define a run-time parameter in a called subprocedure. Each parameter requires its own DEFINE statement. The parameters must be specified in the RUN statement in the same order as defined with DEFINE statements.
- Progress disregards an empty pair of braces ({}).
- The maximum length of the arguments you can pass to an include file is determined by the Input Characters (-inp) startup parameter.
- An argument *argument-name* behaves like a scoped preprocessor name. Thus, if you define a preprocessor name, *argument-name*, its value replaces the value of any argument *argument-name* passed to the same file at the point where the preprocessor name, *argument-name*, is defined.

See also

; Special character, { } Include file reference, { } Preprocessor name reference, COMPILE statement, DEFINE PARAMETER statement, RUN statement

{ } Include file reference

Causes Progress to retrieve the statements in a file and compile them as part of the main procedure if it encounters the file's name inside braces ({ }) when compiling the procedure. You can name arguments you want substituted in the file before compilation.

Syntax

```
{ include-file
  [ argument | { &argument-name = "argument-value" } ] ... }
```

Enter the braces ({ }) as shown; they do not represent syntax notation in this description.

include-file

The name of an external operating system file that contains statements you want included during the compilation of a procedure. This filename follows normal operating system naming conventions and is case sensitive on UNIX. If the file you name has an unqualified path name, Progress searches directories based on the PROPATH environment variable.

argument

A value used by *include-file*. When Progress compiles the main procedure (the procedure containing the braces), it copies the contents of *include-file* into that procedure, substituting any *arguments*. The first argument replaces {1} in the included file, the second argument replaces {2}, etc. So, you can use included procedures with arguments even when you precompile a procedure.

&argument-name = "argument-value"

The *argument-name* is the name of the argument you want to pass to the include file. You can use variable names, field names, and reserved words as argument names.

The *argument-value* is the value of the argument you pass to the include file. Enclose the *argument-value* in quotation marks.

Examples

The main procedure uses externally defined and maintained files for the layout and display of a customer report. You can use these same include files in many procedures.

r-inc1.p

```
FOR EACH customer:
  {r-fcust.i}
  {r-dcust.i}
END.
```

r-fcust.i

```
FORM customer.cust-num customer.name LABEL "customer Name"
  customer.phone FORMAT "999-999-9999".
```

r-dcust.i

```
DISPLAY customer.cust-num customer.name customer.phone.
```

The following example references an include file that can take up to five arguments, and the main routine passes four arguments:

r-incl2.p

```
DEFINE VARIABLE var1 as INTEGER INITIAL 9.
DEFINE VARIABLE var2 as DECIMAL INITIAL 6.43.
DEFINE VARIABLE var3 as LOGICAL INITIAL TRUE.
    /* any statements */
{r-show.i point-A var1 var2 var3}
    /* any statements */
```

When the main procedure is compiled, the line referencing the show include file is replaced by the following line:

r-show.i

```
MESSAGE "At" "{1}" "{2}" {2} "{3}" {3} "{4}" {4} "{5}" {5}.
```

This example shows how you can use include files to extend the Progress language. The main procedure uses a new statement, `r-show.i`, to display the values of fields or variables at various points in a procedure. The include file in this example can handle up to five passed arguments. The main procedure only passes four (point-A, var1, var2, and var3). The output for this example is as follows:

```
MESSAGE At point-A var1 9 var2 6.43 var3 yes
```

The r-custin.p procedure displays a frame for each customer that you can update with customer information. The procedure includes r-cstord.i and passes the named argument &frame-options, and the value of the argument (CENTERED ROW 3 NO-LABEL) to the include file. When the include file references the &frame-options argument, it uses the value of the argument, and therefore displays the OVERLAY frame cust-ord as a centered frame at row 3 without a label, as shown:

r-custin.p

```
FOR EACH customer:
  {r-cstord.i &frame-options = "CENTERED ROW 3 NO-LABEL"}.
  UPDATE customer.cust-num name address address2 city state postal-code
    phone credit-limit WITH FRAME cust-ord.
END.
```

r-cstord.i

```
FORM   "Cust #" AT 1  customer.cust-num AT 10 SKIP(1)
  customer.name AT 10
  customer.address AT 10
  customer.address2 AT 10
  customer.city AT 10  customer.city customer.state
  customer.postal-code SKIP(1)
  "Phone " AT 1 customer.phone FORMAT "999/999-9999" AT 10
  "Max Crd" AT 1 customer.credit-limit AT 10
  WITH FRAME cust-ord OVERLAY {&frame-options}.
```

Include files are particularly useful for using form layouts in multiple procedures, especially if you do not include the keyword FORM or the closing period (.) of the FORM statement.

r-cust.f

```
customer.cust-num
customer.name SKIP(2)
customer.state
```

The r-incl3.p procedure includes the r-cust.f file as the definition of a FORM statement, as shown:

r-incl3.p

```
FORM {r-cust.f}.
```

The r-incl4.p procedure uses the include file as a layout for a DISPLAY statement, as shown:

r-incl4.p

```
FOR EACH customer:
  DISPLAY {r-cust.f} WITH 3 DOWN.
END.
```

Notes

- When you use braces to include one procedure in another, Progress does not include the second procedure until it compiles the first one. This technique has the same effect as using the Editor to copy statements into the main procedure. At times, separate include files are easier to maintain.
- You can nest include files. (They can contain references to other include files.) The number of nested include files is limited by the number of file descriptors available on the system.
- If you have many nested include files and you are running on a Sequent machine, use Maximum Files (-Mv) startup parameter to control the number of files you can open simultaneously.
- When you have a base procedure and want to make several copies of it, changing it slightly each time, use include files with parameters. For example, at times you might only want to change the name of some files or fields used by the procedure.
- If you define a preprocessor name and later pass a compile-time argument with the same name, but a different value, to a procedure or include file, the value of the initial preprocessor name remains unchanged. Thus, a compile-time argument is scoped to the file to which it is passed.
- Instead of maintaining duplicate source files, create a single include file with the variable portions (such as the names of files and fields) replaced by {1}, {2}, etc. Then each procedure you write can use that include file, passing file and field names as arguments.

- You can use the name of an include file as an argument to another include file. For example, a reference to {{1}} in an include file causes Progress to include the statements from the file with the name that passed as the first argument.
- Progress disregards an empty pair of braces ({}).
- If you use double quotes (“ ”) around arguments in an argument list, Progress removes them. However, if you use single quotes (’ ’), Progress passes them. To pass one set of double quotes, you must use four sets of double quotes.
- When Progress reads an include file into the source, it appends a space character to the end of an include file. For example, the following include file `r-string.i` contains data that is used by `r-incstr.p`:

  r-string.i

  ```
  abcde
  ```

  r-incstr.p

  ```
  DISPLAY LENGTH("{r-string.i}").
  ```

  Although `r-string.i` contains five letters, when you run `r-incstr.p`, it returns the value 6 because Progress appends a space character to the end of `r-string.i`.
- The maximum length of the arguments you can pass to an include file is determined by the Input Characters (-inp) startup parameter.

See also { } Argument reference, { } Preprocessor name reference, COMPILE statement, DEFINE PARAMETER statement, RUN statement

{ } Preprocessor name reference

References the value of a preprocessor name in any 4GL or preprocessor expression.

Syntax

```
{ &preprocessor-name }
```

Enter the braces ({}) as shown; they do not represent syntax notation in this description.

&preprocessor-name

Expands the name, *preprocessor-name*, to its defined value. You can define preprocessor names using either the &GLOBAL-DEFINE preprocessor directive or the &SCOPED-DEFINE preprocessor directive. Progress also provides a set of built-in preprocessor names that you can reference for a variety of session information. Table 3 lists each built-in preprocessor name with its description.

Table 3: Built-in preprocessor names *(1 of 2)*

The preprocessor name ...	Expands to an unquoted string ...
BATCH-MODE	Equal to "yes" if the Batch (-b) startup parameter was used to start the client session. Otherwise, it expands to "no".
FILE-NAME	That contains the name of the file being compiled.[1] If you want only the name of the file as specified in the { } Include File Reference, the RUN statement, or the COMPILE statement, use the argument reference {0}.
LINE-NUMBER	That contains the current line number in the file being compiled. If you place this reference in an include file, the line number is calculated from the beginning of the include file.
OPSYS	That contains the name of the operating system on which the file is being compiled. The OPSYS name can have the same values as the OPSYS function. The possible values are “UNIX” and “WIN32".[2]

Table 3: Built-in preprocessor names *(2 of 2)*

The preprocessor name ...	Expands to an unquoted string ...
SEQUENCE	Representing a unique integer value that is sequentially generated each time the SEQUENCE preprocessor name is referenced. When a compilation begins, the value of {&SEQUENCE} is 0; each time {&SEQUENCE} is referenced, the value increases by 1. To store the value of a reference to SEQUENCE, you must define another preprocessor name as {&SEQUENCE} at the point in your code you want the value retained.
WINDOW-SYSTEM	That contains the name of the windowing system in which the file is being compiled. The possible values include "MS-WINDOWS", “MS-WIN95", and ”TTY".[3]

[1] When running the source code of a procedure file loaded into the Procedure Editor or the AppBuilder, {&FILE-NAME} expands to a temporary filename, not the name of the file under which the source code might be saved.

[2] Progress supports an override option that enables applications that need to return the value of MS-DOS for all Microsoft® operating systems to do so. For example, if you do not want the value WIN32 to be returned when either Windows 95 or Windows NT operating systems are recognized, you can override this return value by defining the Opsys key in the Startup section of the current environment, which can be in the registry or in an initialization file. If the Opsys key is located, the OPSYS function returns the value associated with the Opsys key on all platforms.

[3] Progress supports an override option for the &WINDOW-SYSTEM preprocessor name that provides backward compatibility. This option enables applications that need the WINDOW-SYSTEM preprocessor name to return the value of MS-WINDOWS for all Microsoft operating systems to do so. To establish this override value, define the `WindowSystem` key in the Startup section of the current environment, which can be in the registry or in an initialization file. If the `WindowSystem` key is located, the WINDOW-SYSTEM preprocessor name returns the value associated with the `WindowSystem` key on all platforms.

Table 4 lists the additional built-in preprocessor names that apply to SpeedScript.

Table 4: SpeedScript built-in preprocessor names

The preprocessor name ...	Expands to an unquoted string ...
DISPLAY	DISPLAY {&WEBSTREAM}
OUT	PUT {&WEBSTREAM} UNFORMATTED
OUT-FMT	PUT {&WEBSTREAM} FORMATTED
OUT-LONG	EXPORT {&WEBSTREAM}
WEBSTREAM	STREAM WebStream

Examples

The following procedure `r-prprc1.p` shows how you can reference a built-in preprocessor name and include it in a character string:

r-prprc1.p

```
MESSAGE "The current operating system is" "{&OPSYS}."
    VIEW-AS ALERT-BOX.
```

The procedure r-prprc2.p shows how to capture the value of a {&SEQUENCE} reference. In this example, {&SEQUENCE} is referenced three times, once each to assign its value to wvar (0) and xvar (1) at run time. The third reference defines the preprocessor name Last-Value with the value 3. As shown, Last-Value is assigned unchanged to both yvar and zvar, each of which take the value 3 at run time:

r-prprc2.p

```
DEFINE VARIABLE wvar AS INTEGER.
DEFINE VARIABLE xvar AS INTEGER.
DEFINE VARIABLE yvar AS INTEGER.
DEFINE VARIABLE zvar AS INTEGER.

wvar = {&SEQUENCE}.
xvar = {&SEQUENCE}.

&GLOBAL-DEFINE Last-Value {&SEQUENCE}

yvar = {&Last-Value}.
zvar = {&Last-Value}.

MESSAGE "wvar =" wvar SKIP "xvar =" xvar SKIP
        "yvar =" yvar SKIP "zvar =" zvar VIEW-AS ALERT-BOX.
```

The procedure r-prprc3.p shows how preprocessor names override compile-time arguments. In this example, r-prprc3.p defines the preprocessor name My-Name as "Daniel". It then passes the compile-time argument My-Name, with the value "David", to the include file r-prprc3.i, which in turn defines a preprocessor name My-Name as "Donald".

r-prprc3.p

```
&SCOPED-DEFINE My-Name "Daniel"
{r-prprc3.i &My-Name = "David"}
MESSAGE "My-Name preprocessed in r-prprc3.p is" {&My-Name} + "."
    VIEW-AS ALERT-BOX.
```

r-prprc3.i

```
MESSAGE "My-Name argument in r-prprc3.i is" "{&My-Name}" + "."
    VIEW-AS ALERT-BOX.
&SCOPED-DEFINE My-Name "Donald"
MESSAGE "My-Name preprocessed in r-prprc3.i is" {&My-Name} + "."
    VIEW-AS ALERT-BOX
```

During execution, the first message included by r-prprc3.i displays the value of the My-Name argument, "David". The second message included by r-prprc3.i displays the value of the following My-Name preprocessor name, defined as "Donald", permanently overriding "David" passed by the My-Name argument. Finally, the message in r-prprc3.p displays the value of the My-Name preprocessor name that was initially defined there, "Daniel", because the value from My-Name established in r-prprc3.i ("Donald") went out of scope during compilation.

Note also that the reference to the My-Name compile-time argument in r-prprc3.i is inside double-quotes, because Progress passes string constant values for compile-time arguments without the surrounding double-quotes.

You can encounter compilation problems mixing preprocessor names with compile-time argument names. The following example, a variation of r-prprc3.i, does not compile, even when passed a My-Name argument as an include file. This is because the preprocessor My-Name value overrides the argument My-Name value, as shown:

```
&SCOPED-DEFINE My-Name "Donald"
MESSAGE "My-Name preprocessed in r-prprc3.i is" {&My-Name} + "."
    VIEW-AS ALERT-BOX.
MESSAGE "My-Name argument in r-prprc3.i is" "{&My-Name}" + "."
    VIEW-AS ALERT-BOX.
```

Because the preprocessor My-Name defines a quoted "Donald" value, Progress replaces "{&My-Name}" in the fourth line with ""Donald"". This appears to the compiler as two empty strings and an unknown variable reference (Donald). Although you can do it with care, in general, avoid using the same names for compile-time arguments and preprocessor names.

Notes

- Progress expands preprocessor names wherever and in whatever context it finds them, including inside quoted character strings.
- If you define a preprocessor name in the same file and with the same name as a compile-time argument passed to the file, the value of the preprocessor name takes precedence over the value of the argument name from the point where the preprocessor name is defined.

See also

&GLOBAL-DEFINE preprocessor directive, &SCOPED-DEFINE preprocessor directive, { } Argument reference, { } Include file reference, ; Special character

&GLOBAL-DEFINE preprocessor directive

Globally defines a compile-time constant (preprocessor name).

Syntax

```
&GLOBAL-DEFINE preprocessor-name definition
```

preprocessor-name

The preprocessor name (compile-time constant) that you supply. Progress reserved keywords are allowed, but cannot be used in preprocessor expressions.

definition

A string of characters (or preprocessor references that evaluate to a string of characters) whose content the preprocessor substitutes for *preprocessor-name* during compilation. If the definition is longer than one line, a tilde (~) at the end of the first line indicates continuation to the next line.

Examples

In this example, the preprocessor name MAX-EXPENSE is defined as the text string "5000":

```
&GLOBAL-DEFINE MAX-EXPENSE 5000
```

Wherever the reference {&MAX-EXPENSE} appears in the source code, the preprocessor substitutes the text string “5000". For example, the preprocessor changes this line of code:

```
IF tot-amount <= {&MAX-EXPENSE} THEN DO:
```

to this line:

```
IF tot-amount <= 5000 THEN DO:
```

Notes

- You must place the &GLOBAL-DEFINE directive at the beginning of a line, preceded only by blanks, tab characters, or comments (`/* comment */`). The preprocessor trims all leading and trailing spaces from *definition*.
- The syntax of the &GLOBAL-DEFINE and &SCOPED-DEFINE directives are identical but these directives are used differently.

See also

{ } Preprocessor name reference, &SCOPED-DEFINE preprocessor directive, &UNDEFINE preprocessor directive, DEFINED preprocessor function

&IF, &THEN, &ELSEIF, &ELSE, and &ENDIF preprocessor directives

These directives set logical conditions for the inclusion of blocks of code to compile.

Syntax

```
&IF expression &THEN
  block
[ &ELSEIF expression &THEN
  block ] ...
[ ELSE
  block ]
&ENDIF
```

expression

An expression that can contain preprocessor name references, the operators listed in Table 6, the Progress functions listed in Table 7, and the DEFINED( ) preprocessor function.

When it encounters an &IF directive, the preprocessor evaluates the expression that immediately follows. This expression can continue for more than one line; the &THEN directive indicates the end of the expression. If the expression evaluates to TRUE, then the block of code between it and the next &ELSEIF, &ELSE, or &ENDIF is compiled. If the expression evaluates to FALSE, the block of code is not compiled and the preprocessor proceeds to the next &ELSEIF, &ELSE, or &ENDIF directive. No include files referenced in this block of code are included in the final source. You can nest &IF directives.

The expression that follows the &ELSEIF directive is evaluated only if the &IF expression tests false. If the &ELSEIF expression tests TRUE, the block of code between it and the next &ELSEIF, &ELSE, or &ENDIF directive is compiled. If the &ELSEIF expression tests FALSE, the preprocessor proceeds to the next &ELSEIF, &ELSE, or &ENDIF directive.

The block of code between the &ELSE and &ENDIF directives is compiled only if the &IF expression and the &ELSEIF expressions all test false. If there are no &ELSEIF directives, the block of code is compiled if the &IF expression tests false.

Once any &IF or &ELSEIF expression evaluates to TRUE, no other block of code within the &IF...&ENDIF block is compiled.

The &ENDIF directive indicates the end of the conditional tests and the end of the final block of code to compile.

Table 5 shows how preprocessor expressions are evaluated.

Table 5: Preprocessor expressions

Type of expression	TRUE	FALSE
LOGICAL	TRUE	FALSE
CHARACTER	non-empty	empty
INTEGER	non-zero	0
DECIMAL	not supported	not supported

Table 6 lists the operators supported within preprocessor expressions. These operators have the same precedence as the regular Progress 4GL operators.

Table 6: Preprocessor operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
=	Equality
<>	Inequality
>	Greater than
<	Less than
=>	Greater than or equal to
<=	Less than or equal to
AND	Logical and
OR	Logical or
NOT	Logical not
BEGINS	Compares the beginning letters of two expressions
MATCHES	Compares two strings

Table 7 lists the Progress 4GL functions supported within preprocessor expressions.

Table 7: Functions allowed in preprocessor expressions

ABSOLUTE	ASC	AUDIT-ENABLED
DATE	DAY	DBTYPE
DECIMAL	ENCODE	ENTRY
ETIME	EXP	FILL
INDEX	INTEGER	KEYWORD
KEYWORDALL	LC	LEFT-TRIM
LENGTH	LIBRARY	LOG
LOOKUP	MATCHES	MAXIMUM
MEMBER	MINIMUM	MODULO
MONTH	NUM-ENTRIES	OPSYS
PROPATH	PROVERSION	RANDOM
REPLACE	RIGHT-TRIM	R-INDEX
ROUND	SQRT	STRING
SUBSTITUTE	SUBSTRING	TIME
TODAY	TRIM	TRUNCATE
WEEKDAY	YEAR	

Note When the preprocessor evaluates expressions, all arithmetic operations are performed with 32-bit integers. Preprocessor name references used in arithmetic operations must evaluate to integers.

See also &GLOBAL-DEFINE preprocessor directive, &SCOPED-DEFINE preprocessor directive, &UNDEFINE preprocessor directive

&MESSAGE preprocessor directive

Displays a message at compile time in the Compiler Messages dialog box.

Note: Does not apply to SpeedScript programming.

Syntax

```
&MESSAGE text-string
```

text-string

A string of characters, preprocessor name references, named include file arguments, or any combination of these that results in a character string to display. The *text-string* argument does not need to be quoted.

Examples

This is a possible compile-time message directive:

```
&MESSAGE Compiling the {&FILE-NAME} file.
           .
           .
           .
```

If this fragment appears in a procedure file, `cmessage.p`, compiling this file with the COMPILE statement causes the following message to be included with the compiler messages:

```
Compiling the cmessage.p file.
```

See also

{ } Preprocessor name reference

&SCOPED-DEFINE preprocessor directive

Defines a compile-time constant (preprocessor name) non-globally.

Syntax

```
&SCOPED-DEFINE preprocessor-name definition
```

preprocessor-name

The preprocessor name (compile-time constant) that you supply. Progress reserved keywords are allowed, but cannot be used in preprocessor expressions.

definition

A string of characters (or preprocessor references that evaluate to a string of characters) whose content the preprocessor substitutes for *preprocessor-name* during compilation. If definition is longer than on line, a tilde (~) at the end of the first line indicates continuation to the next line.

Notes

- You must place the &SCOPED-DEFINE directive at the beginning of a line, preceded only by blanks, tab characters, or comments (`/* comment */`). The preprocessor trims all leading and trailing spaces from *definition*.
- The syntax of the &GLOBAL-DEFINE and &SCOPED-DEFINE directives are identical but these directives are used differently.

See also

{ } Preprocessor name reference, &GLOBAL-DEFINE preprocessor directive, &UNDEFINE preprocessor directive, DEFINED preprocessor function

&UNDEFINE preprocessor directive

Undefines a compile-time constant (preprocessor name).

Syntax

```
&UNDEFINE preprocessor-name
```

preprocessor-name

The preprocessor name (compile-time constant) that you want to undefine.

Notes

- When you use the &UNDEFINE directive, Progress warns you if the name you want to undefine was not previously defined.
- The &UNDEFINE directive undefines the currently active name. It also undefines named include file arguments.
- To globally define the same name more than once, use this directive to undefine the name before redefining it. If you do not undefine the global name before redefining it, the compiler produces a warning message. This does not apply to non-globally (scoped) defined names.

See also

&GLOBAL-DEFINE preprocessor directive, &SCOPED-DEFINE preprocessor directive, DEFINED preprocessor function

/* Comments */

Allows you to add explanatory text to a procedure between the /* and */ characters.

Syntax

```
/* comment */
```

comment

Descriptive text.

Note: Comments can be nested.

Examples

This example uses comments to document the history of procedure modifications:

r-comm.p

```
/* Procedure written  9/5/87 by CHC
             revised  9/27/87 by DG */

FOR EACH customer:
    DISPLAY cust-num name contact phone.
END.
```

The following example uses comments to describe what the procedure does:

r-comm2.p

```
/* step through unshipped orders */
FOR EACH order WHERE ship-date = ?:
  /* display order date, promise date, terms */
  DISPLAY order-date promise-date terms.
  /*
    FOR EACH order-line OF order:
      /* display all order-lines of each order */
      DISPLAY order-line.
    END.
  */
END.
```

The comment symbols that enclose the inner FOR EACH block turn that block into a comment for testing purposes. Since you can nest comments, Progress correctly processes any comments already in the bypassed code.

+ Unary positive operator

Preserves the positive or negative value of a numeric expression. Do not confuse this operator with the addition operator that you use to add expressions together.

Syntax

```
+ expression
```

expression

An expression whose value is numeric.

Example

In this example, the sign of credit-limit is preserved as is the sign of the sum of credit-limit + 100. The unary positive is not necessary; it is used simply to document the procedure.

r-unpos.p

```
DEFINE VARIABLE old-max LIKE credit-limit LABEL "Old Limit".

FOR EACH customer:
  old-max =+ credit-limit.
  credit-limit =+(credit-limit + 100).
  DISPLAY name old-max credit-limit.
END.
```

+ Addition operator

Adds two numeric expressions.

Syntax

```
expression + expression
```

expression

An expression whose value is numeric.

Example

In the following example, the addition operator (+) adds 100 to the value of the credit-limit field:

r-addn.p

```
FOR EACH customer:
  credit-limit = credit-limit + 100.
END.
```

Note

Adding two decimal expressions produces a decimal value. Adding two integer expressions produces an integer value. Adding an integer expression and a decimal expression produces a decimal value.

+ Concatenation operator

Produces a character value by joining two character strings or expressions.

Syntax

```
expression + expression
```

expression

An expression whose value is a character string. If any expression is a LONGCHAR, the result is a LONGCHAR. Also, the result converts to the code page of the expression on the left.

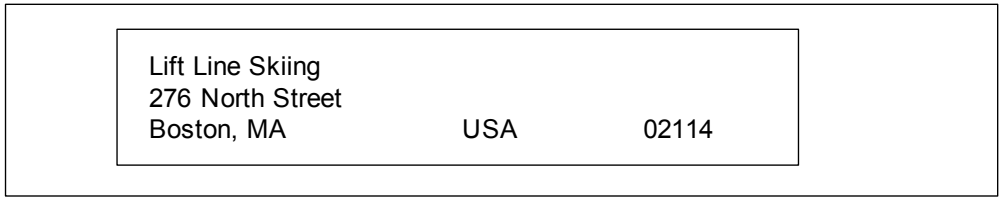
Example

This procedure prints mailing labels. It uses the concatenation operator (+) to ensure that the third line of each label shows the city and state separated by a comma and a space. The FORMAT x(16) is specified to provide room for up to 16 characters in the result of the concatenation. If a FORMAT is not given, then Progress only displays the first eight characters of the result since x(8) is the default format for a character expression.

r-conc.p

```
FOR EACH customer:
  DISPLAY  SKIP(1) name SKIP address SKIP
    city + ", " + state FORMAT "x(16)" country postal-code SKIP(2).
END.
```

This is a label produced by this procedure:



Note

If **any** of the string values you concatenate is the Unknown value (?), then the result is the Unknown value (?). This might lead to unexpected results if a field used in an expression is not mandatory. For example, you might have fields for a person's first name, last name, and middle initial. You might combine these into a full name with an expression like the following:

```
DISPLAY fname + " " + minit + " " + lname FORMAT "x(36)".
```

If `minit` is not a mandatory field, it might be set to the Unknown value (?) in some records. If so, then those records are displayed as the Unknown value (?). You can avoid this by using conditional code. For example:

```
DISPLAY fname + " " + (IF minit <> ? THEN minit + ". " ELSE "") +
" " + lname FORMAT "x(36)".
```

+ Date addition operator

Adds a number of days to a date, producing a date result.

Syntax

```
date + days
```

date

An expression that evaluates to a DATE value.

days

An expression with a value of the number of days you want to add to a *date*.

Example

This procedure finds all unshipped orders that are at least one week overdue. If the order is not shipped and the promised date is more than seven days ago, the procedure finds the record for the customer who placed the order and displays the order and customer data.

r-dadd.p

```
DISPLAY "ORDERS SCHEDULED TO SHIP MORE THAN ONE WEEK LATE".
FOR EACH order WHERE ship-date = ?:
  IF TODAY > (promise-date + 7)
  THEN DO:
    FIND customer OF order.
    DISPLAY order.order-num order.cust-num customer.name promise-date
    customer.terms.
  END.
END.
```

Notes

- The date addition operator rounds days to the nearest integer value.
- To add a specific number of days or a specific number of milliseconds to a DATETIME, use the DATETIME function. For example:

```
new-datetime = DATETIME( DATE(old-datetime) + days,
   MTIME(old-datetime) + milliseconds ).
```

  The DATETIME function ensures that the time portion remains within the valid range, by adding day(s) to the date part when the time part goes over the number of milliseconds in a day.

- To add a specific number of days or milliseconds to a DATETIME-TZ, use the DATETIME-TZ function. For example:

```
new-datetime-tz = DATETIME-TZ( DATE(old-datetime-tz) + days,
   MTIME (old-datetime-tz) + milliseconds, TIMEZONE(old-dateime-tz)).
```

  The DATETIME-TZ function ensures that the time portion remains within the valid range, by adding day(s) to the date portion when the time part goes over the number of milliseconds in a day.

– Unary negative operator

Reverses the sign of a numeric expression. Do not confuse this operator with the subtraction operator that subtracts one expression from another.

Syntax

```
- expression
```

expression

An expression whose value is numeric.

Example

If you supply a negative value for the variable x, the following example procedure uses the unary negative operator (-) to reverse the sign of x, producing the absolute value of x (abs-x):

r-uneg.p

```
DEFINE VARIABLE x AS DECIMAL LABEL "X".
DEFINE VARIABLE abs-x AS DECIMAL LABEL "ABS(X)".

REPEAT:
  SET x.
  IF x < 0
  THEN abs-x = -x
  ELSE abs-x = x.
  DISPLAY abs-x.
END.
```

– Subtraction operator

Subtracts one numeric expression from another numeric expression.

Syntax

```
expression - expression
```

expression

An expression with a numeric value.

Example

This procedure determines the amount of inventory available by subtracting the amount allocated from the total on hand:

r-subt.p

```
DEFINE VARIABLE free-stock LIKE on-hand LABEL "Free Stock".

FOR EACH item:
  free-stock = on-hand - allocated.
  DISPLAY item-num item-name on-hand allocated free-stock.
END.
```

Note

Subtracting one decimal expression from another produces a decimal value. Subtracting one integer expression from another produces an integer. Subtracting an integer expression from a decimal expression (or subtracting a decimal expression from an integer expression) produces a decimal value.

– Date subtraction operator

Subtracts a number of days from a date to produce a date result, or subtracts one date from another to produce an integer result that represents the number of days between the two dates.

Syntax

```
date - { days | date }
```

date

An expression that evaluates to a DATE value.

days

An expression with a value of the number of days you want to subtract from a *date*.

Example

This procedure finds all unshipped orders. If the promised date is more than one week ago, the procedure finds the customers who placed the order and displays the order and customer data.

r-dsub.p

```
DISPLAY "ORDERS SCHEDULED TO SHIP MORE THAN ONE WEEK LATE".
FOR EACH order WHERE ship-date = ?:
  IF (TODAY - 7) > promise-date
  THEN DISPLAY order.order-num order.cust-num promise-date
    (TODAY - promise-date) LABEL "Days Late".
END.
```

Notes

- The date subtraction operator rounds days to the nearest integer value.
- To get the number of days between two DATETIME or DATETIME-TZ variable values, use the DATE function. For example:

```
num-days = DATE(dt2) - DATE(dt1)
```

  This operation does not take the time portion into account.

  To ensure the correct result when working with two DATETIME-TZ values, convert one of the values to the time zone of the other. For example:

```
temp-dttz = dt1.
TIMEZONE(temp-dttz) = TIMEZONE(dt2).
num-days = DATE(dt2) – DATE(temp-dttz).
```

- To subtract a specific number of days or milliseconds from a DATETIME, use the DATETIME function. For example:

```
new-datetime = DATETIME( DATE(old-datetime) - days,
   MTIME (old-datetime) - milliseconds).
```

  The DATETIME function ensures the time portion remains within a valid range by borrowing a day from the date portion, when necessary.

- To subtract a specific number of days or milliseconds from a DATETIME-TZ, use the DATETIME-TZ function. For example:

```
new-datetime-tz = DATETIME-TZ( DATE(old-datetime-tz) - days,
   MTIME (old-datetime-tz) – milliseconds,TIMEZONE(old-dateime-tz)).
```

  The DATETIME-TZ function ensures the time portion remains within a valid range by borrowing a day from the date portion, when necessary.

* Multiplication operator

Multiplies two numeric expressions.

Syntax

```
expression * expression
```

expression

An expression with a numeric value.

Example

This procedure computes the value of the on-hand inventory for each item. If the on-hand inventory is negative, the procedure sets the inventory value to 0.

r-mult.p

```
DEFINE VARIABLE inv-value AS DECIMAL LABEL "VALUE".

FOR EACH item:
    inv-value = on-hand * price.
    IF inv-value < 0
    THEN inv-value = 0.
    DISPLAY item.item-num item-name on-hand price inv-value.
END.
```

Note

Multiplying two decimal expressions produces a decimal value. Multiplying two integer expressions produces an integer value. Multiplying an integer expression and a decimal expression produces a decimal value.

/ Division operator

Divides one numeric expression by another numeric expression, producing a decimal result.

Syntax

```
expression / expression
```

expression

An expression that evaluates to a numeric value.

Example

This procedure divides the number of items allocated by the number of items on hand, producing a decimal value. The multiplication operator (*) converts that decimal value to a percentage.

r-div.p

```
DISPLAY "INVENTORY COMMITMENTS AS A PERCENT OF UNITS ON HAND".

FOR EACH item:
  DISPLAY item.item-num item-name alloc on-hand (alloc / on-hand) * 100
    FORMAT ">>9" LABEL "PCT".
END.
```

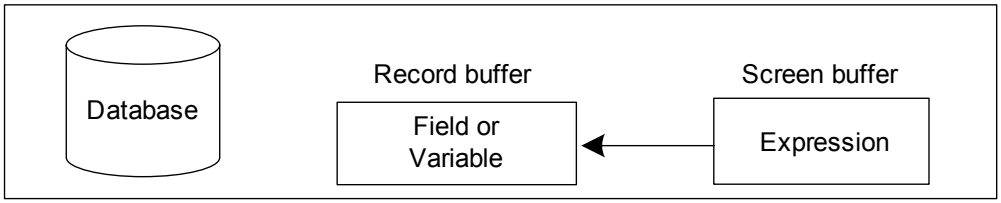
Notes

- Progress **always** performs division as a decimal operation (the product of 5 / 2 is 2.5, not 2). If you assign the result to an integer field, Progress rounds the decimal to make the assignment. When you want Progress to truncate a quotient to an integer, use the TRUNCATE function (TRUNCATE(5 / 2, 0) is 2).
- The result of dividing a number by 0 is the Unknown value (?), and Progress does not display an error message.

= Assignment operator

Assigns the value of an expression to a database field or variable.

Data movement



Syntax

```
field = expression [ NO-ERROR ]
```

field

The name of a database field or variable to which you want to assign the value of the expression. If the field is an array, and you do not name a particular element, Progress stores *expression* in each element of the array. If you name a particular element, Progress stores *expression* in that element.

The left side of an assignment can also be an attribute or Progress keyword (such as ENTRY, FRAME-VALUE, SUBSTRING, or OVERLAY).

expression

An expression with a data type that is consistent with the data type of the *field*. If *field* is integer and *expression* is decimal, then Progress rounds the value of the expression before assigning it. If *field* is decimal and *expression* is decimal, then Progress rounds the value of the expression to the number of decimal places defined for the field in the Dictionary or defined or implied for a variable.

NO-ERROR

Suppresses any errors that occur as a result of the operation. After the assignment completes, you can check the ERROR-STATUS system handle for information about any errors that might have occurred.

If you do not specify this option and an error occurs, the assignment is canceled and any changes to field values within the assignment are undone. If the assignment occurs within a transaction, any changes to variables, work table fields, and temporary table fields are also undone, unless you define the variable or field with the NO-UNDO option.

Example

This procedure resets all the monthly quota values to 0 in all salesrep records. If you want to set values for individual array elements, you can do so by making an explicit assignment using the assignment statement and a specific array reference, such as month-quota[1] or month-quota[i].

r-asgmnt.p

```
DEFINE VARIABLE ctr   AS INTEGER.

FOR EACH salesrep:
  DO ctr = 1 TO 12:
    salesrep.month-quota = 2500.
  END.
END.
```

Notes

- If you assign a value to a database field, any ASSIGN trigger associated with that field executes at the end of the assignment statement (after any index changes are made). If the trigger returns ERROR, the assignment fails and the database changes are undone.
- You can embed an assignment in a SET or UPDATE statement.
- For multiple assignments, use the ASSIGN statement. This is more efficient than multiple assignment statements.

- You can assign DATE, DATETIME, and DATETIME-TZ data. When the data type expression on the left side of the assignment statement contains more information than the data type expression on the right side provides (for example, *datetime-tz = date* where a DATETIME-TZ value contains more information than a DATE value), the time value defaults to midnight and the time zone value defaults to the session's time zone. When the data type expression on the left side of the assignment statement contains less information than the data type expression on the right side provides (for example, *date = datetime-tz* where a DATE value contains less information than a DATETIME-TZ value), Progress converts the DATETIME-TZ value to the local date and time of the session, then drops the time and time zone.

- You can assign large object data from one BLOB or MEMPTR to another, and from one CLOB, LONGCHAR, or CHARACTER to another. You cannot assign large object data between BLOBs and CLOBs or MEMPTRs and LONGCHARs. You can accomplish this, indirectly, by using the COPY-LOB statement. For more information, see the COPY-LOB statement reference entry.

Note: When assigning BLOB or CLOB fields, the field must appear by itself on either the right-hand or the left-hand side of the assignment.

Table 8 lists the default character conversions Progress performs when assigning CLOB, LONGCHAR, and CHARACTER data between a source and target object. References to CLOBCP and CLOBDB represent CLOB data in either the CLOB's defined code page or the database's defined code page, respectively. References to the "fixed code page" represent the code page of a target LONGCHAR variable set using the FIX-CODEPAGE function.

Table 8: Default Assignment operator character conversions *(1 of 2)*

When the target object (on the left) is a ...	And the source object (on the right) is a ...	Progress converts the data in the source object to ...
LONGCHAR	CLOBDB	`-cpinternal` or the fixed code page.
LONGCHAR	CLOBCP	The CLOB's defined code page or the fixed code page.
LONGCHAR	CHARACTER	`-cpinternal` or the fixed code page.
CLOBDB	CHARACTER	The database's defined code page.

Table 8: Default Assignment operator character conversions *(2 of 2)*

When the target object (on the left) is a ...	And the source object (on the right) is a ...	Progress converts the data in the source object to ...
CLOBDB	LONGCHAR	The database's defined code page.
CLOBCP	CHARACTER	The CLOB's defined code page.
CLOBCP	LONGCHAR	The CLOB's defined code page.
CHARACTER	CLOBDB	`-cpinternal` code page.
CHARACTER	CLOBCP	`-cpinternal` code page.
CHARACTER	LONGCHAR	`-cpinternal` code page.

- When you assign the Unknown value (?) to a BLOB or CLOB field, Progress deletes any associated object data.

See also ASSIGN statement, Data types, COPY-LOB statement, FIX-CODEPAGE function

ABSOLUTE function

Returns the absolute value of a numeric value.

Syntax

```
ABSOLUTE ( n )
```

n

An integer or decimal expression. The return value is the same format as *n*.

Example

This procedure calculates the number of miles you drive between highway exit ramps:

r-abs.p

```
DEFINE VARIABLE mark-start  AS DECIMAL NO-UNDO.
DEFINE VARIABLE mark-finish AS DECIMAL NO-UNDO.
DEFINE VARIABLE units AS LOGICAL FORMAT "miles/kilometers" NO-UNDO.

FORM
  mark-start  LABEL "Mile marker for highway on-ramp" SKIP
  mark-finish LABEL "Mile marker next to your exit" SKIP(1)
  units LABEL "Measure in <m>iles or <k>ilometers" SKIP(1)
  WITH FRAME question SIDE-LABELS
  TITLE "This program calculates distance driven.".

UPDATE mark-start mark-finish units WITH FRAME question.

DISPLAY
  "You have driven" ABSOLUTE(mark-start - mark-finish) units
  WITH NO-LABELS FRAME answer.
```

ACCUM function

Returns the value of an aggregate expression that is calculated by an ACCUMULATE or aggregate phrase of a DISPLAY statement.

Syntax

```
ACCUM aggregate-phrase expression
```

aggregate-phrase

A phrase that identifies the aggregate value it should return. This is the syntax for *aggregate-phrase*:

```
{   AVERAGE
  | COUNT
  | MAXIMUM
  | MINIMUM
  | TOTAL
  | SUB-AVERAGE
  | SUB-COUNT
  | SUB-MAXIMUM
  | SUB-MINIMUM
  | SUB-TOTAL
} [ BY break-group ]
```

For more information on aggregate items, see the Aggregate phrase reference entry.

expression

An expression that was used in an earlier ACCUMULATE or DISPLAY statement. The expression you use in the ACCUMULATE or DISPLAY statement and the expression you use in the ACCUM function must be in exactly the same form. (For example, “on-hand * cost” and “cost * on-hand” are not in exactly the same form.) For the AVERAGE, SUB-AVERAGE, TOTAL, and SUB-TOTAL aggregate phrases, *expression* must be numeric.

Example

This procedure shows a total for the extended price of each item on an order. The running total of the order is displayed as well as the order total and grand total for all orders. This procedure accumulates totals at three levels:

r-accum.p

```
FOR EACH order:
  DISPLAY order-num cust-num order-date promise-date ship-date.
  FOR EACH order-line OF order:
    DISPLAY line-num item-num qty price.
    DISPLAY qty * price LABEL "Ext Price".
    ACCUMULATE qty * price (TOTAL).
    DISPLAY (ACCUM TOTAL qty * price) LABEL "Accum Total".
  END.
  DISPLAY (ACCUM TOTAL qty * order-line.price) LABEL "Total".
END.
DISPLAY (ACCUM TOTAL qty * order-line.price) LABEL "Grand Total"
  WITH ROW 1.
```

See also

ACCUMULATE statement, DISPLAY statement

ACCUMULATE statement

Calculates one or more aggregate values of an expression during the iterations of a block. Use the ACCUM function to access the result of this accumulation.

Syntax

```
ACCUMULATE  { expression ( aggregate-phrase ) } ...
```

expression

An expression for which you want to calculate the aggregate value. The expression you use in the ACCUMULATE statement and the expression you use in the ACCUM function (when using the result of the ACCUMULATE statement) must be in exactly the same form. (For example, “A * B” and “B * A” are not in exactly the same form.)

aggregate-phrase

Identifies one or more values to calculate based on a change in expression or a break group. This is the syntax for *aggregate-phrase*:

```
{   AVERAGE
  | COUNT
  | MAXIMUM
  | MINIMUM
  | TOTAL
  | SUB-AVERAGE
  | SUB-COUNT
  | SUB-MAXIMUM
  | SUB-MINIMUM
  | SUB-TOTAL
} ... [ BY break-group ] ...
```

For more information, see the Aggregate phrase reference entry.

Examples

This procedure calculates and displays statistics for all customers, but does not show the detail for each customer:

r-acmlt.p

```
FOR EACH customer:
  ACCUMULATE credit-limit (AVERAGE COUNT MAXIMUM).
END.

DISPLAY "MAX-CREDIT STATISTICS FOR ALL CUSTOMERS:" SKIP(2)
        "AVERAGE =" (ACCUM AVERAGE credit-limit) SKIP(1)
        "MAXIMUM =" (ACCUM MAXIMUM credit-limit) SKIP(1)
        "NUMBER OF CUSTOMERS =" (ACCUM COUNT credit-limit) SKIP(1)
        WITH NO-LABELS.
```

The following procedure lists each item with its inventory value and lists that value as a percentage of the total inventory value of all items; it sorts items by highest value:

r-acmlt2.p

```
FOR EACH item:
  ACCUMULATE on-hand * price (TOTAL).
END.

FOR EACH item BY on-hand * price DESCENDING:
  DISPLAY item-num on-hand price on-hand * price LABEL "Value"
          100 * (on-hand * price) / (ACCUM TOTAL on-hand * price)
          LABEL "Value %".
END.
```

The following procedure displays all customers, sorted by salesrep and country within the list for each salesrep. The procedure calculates the balance for each customer, total balance for each country, and total balance for each salesrep.

r-acc.p

```
FOR EACH customer BREAK BY sales-rep BY country:
   ACCUMULATE balance (TOTAL BY sales-rep BY country).
   DISPLAY sales-rep WHEN FIRST-OF(sales-rep) country name balance.
   IF LAST-OF(country) THEN
      DISPLAY ACCUM TOTAL BY country balance
         COLUMN-LABEL "Country!Total".
   IF LAST-OF(sales-rep) THEN DO:
      DISPLAY sales-rep ACCUM TOTAL BY sales-rep
              balance COLUMN-LABEL "Sales-Rep!Total".
      DOWN 1.
   END.
END.
```

Note You can use the ACCUMULATE statement only in blocks with the implicit looping property. Progress automatically supplies looping services to REPEAT and FOR EACH blocks. See *OpenEdge Development: Progress 4GL Handbook* for more information on block properties.

See also ACCUM function, Aggregate phrase

ADD-INTERVAL function

Adds a time interval to, or subtracts a time interval from, a DATE, DATETIME, or DATETIME-TZ value.

Syntax

```
ADD-INTERVAL (datetime, interval-amount, interval-unit)
```

datetime

An expression whose value is a DATE, DATETIME, or DATETIME-TZ.

interval-amount

A signed integer (positive or negative) indicating the amount of time you want to add to or subtract from *datetime* value.

interval-unit

A character constant, or a character expression that evaluates to one of the following time units: ‘years’, ‘months’, ‘weeks’, ‘days’, 'hours’, ‘minutes’, ‘seconds’ or ‘milliseconds’. These values are case insensitive and may be singular.

Notes

- If *datetime* is a DATE or DATETIME, the time value defaults to midnight and the time zone value defaults to the session's time zone, respectively.
- To add or subtract months or years, this function converts the date to Gregorian before adding or subtracting the year or month value. If the result is an invalid date, the function decrements the day part of the date until a valid date is obtained. For example:
  - Adding 1 month to January 30, 2003 yields February 28, 2003.
  - Adding 13 months to January 30, 2003 yields February 29, 2004 (2004 is a leap year).
  - Subtracting 1 month from December 31, 2003 yields November 30, 2003.

Aggregate phrase

Identifies one or more values to calculate based on a change in an expression or a break group.

Syntax

```
{   AVERAGE
  | COUNT
  | MAXIMUM
  | MINIMUM
  | TOTAL
  | SUB-AVERAGE
  | SUB-COUNT
  | SUB-MAXIMUM
  | SUB-MINIMUM
  | SUB-TOTAL
} ... [ LABEL aggr-label ] [ BY break-group ] ...
```

AVERAGE

Calculates the average of all of the values of the expression in a break group and the average of all of the values of the expression in all break groups.

COUNT

Calculates the number of times the expression was counted in a break group and the count of all the values in all break groups.

MAXIMUM

Calculates the maximum of all of the values of the expression in a break group and the maximum of all the values of the expression in all break groups.

MINIMUM

Calculates the minimum of all of the values of the expression in a break group and the minimum of all the values of the expression in all break groups.

TOTAL

Calculates the subtotal of all of the values of the expression in a break group and the grand total of all of the values of the expression in all break groups. When you use default aggregates, the actual display of the grand total is deferred until the frame goes out of scope.

SUB-AVERAGE

Averages values in a break group. Does not supply an average for all records, just for those in each break group.

SUB-COUNT

Counts the number of times an expression is in a break group. Does not supply a count for all records, just for those in each break group.

SUB-MAXIMUM

Shows the maximum value of an expression in a break group. Does not supply a maximum value for all records, just for those in each break group.

SUB-MINIMUM

Shows the minimum value of an expression in a break group. Does not supply a minimum value for all records, just for those in each break group.

SUB-TOTAL

Subtotals all of the values of the expression in a break group. Does not supply a total value for all records, just for those in each break group.

BY *break-group*

Performs aggregation for break groups if you use the BREAK option in a FOR EACH block header.

LABEL *aggr-label*

Specifies a label for the aggregate value. *aggr-label* is a standard Progress string and can use a string attribute. The string can be translated by Translation Manager II. You can specify a maximum length attribute that is greater than the length of the longest label translation.

Examples

This procedure lists the customer information for all customers (categorized by country) and a subtotal of each country's balance. If you use TOTAL instead of SUB-TOTAL, Progress displays a grand total.

r-aggreg.p

```
FOR EACH customer BREAK BY country:
  DISPLAY name country balance (SUB-TOTAL BY country).
END.
```

In the following procedure, Progress displays the result of the COUNT aggregate even though no accumulation has occurred. In this example, COUNT displays as 0:

r-agcnt.p

```
DEFINE VARIABLE prntr AS LOGICAL INITIAL FALSE.
FOR EACH item:
  IF prntr
  THEN DISPLAY item-name price(COUNT) WITH FRAME pr.
END.
```

In the following procedure, Progress uses "Avg. Credit Limit" and "Max. Credit Limit" as the labels for the AVERAGE and MAXIMUM aggregates respectively:

r-aglim.p

```
FOR EACH customer:
  DISPLAY name credit-limit
    (AVERAGE LABEL "Avg. Credit Limit"
     MAXIMUM LABEL "Max. Credit Limit"
     TOTAL) WITH FRAME frame1 12 DOWN.
 END.
```

Notes

- By default, Progress displays the aggregate result when the aggregate group ends, as long as the block iterates. If you want to suppress automatic display of zero aggregates, use the ACCUMULATE statement to perform the calculation and test the result with the ACCUM function before displaying the result.
- When you use aggregate phrases to accumulate values within shared frames, you must include the ACCUM option in the Frame phrase. See the Frame phrase reference entry for more information.
- An Aggregate phrase is designed to generate aggregate values for blocks that read forward through records in a sequential fashion. In blocks that read records in a non-sequential fashion (for example, FIND PREV, FIND FIRST, FIND LAST, etc.), an aggregate could yield unexpected values.
- Avoid specifying more than one aggregate of the same type for a single field in a block. If an aggregate of the same type for a single field executes more than once during a single iteration of a block, the aggregate could yield unexpected value.
- The BY phrase supports aggregates on break groups. The aggregate for a break group should reside in the block that defines the break group. Avoid positioning the aggregate in a conditional statement or sub-block in the block that defines the break group. Failure to follow these guidelines may yield unexpected values for the aggregate.

  You can build your own algorithms to generate aggregates for break groups in situations that do not adhere to these guidelines. For example, you can use variables to store aggregate values for use in expressions that generate the appropriate aggregate values for break groups across blocks in a procedure.

See also ACCUMULATE statement, FOR statement

ALIAS function

The ALIAS function returns the alias corresponding to the integer value of expression.

Syntax

```
ALIAS ( integer-expression )
```

integer-expression

If there are, for example, three currently defined aliases, the functions ALIAS(1), ALIAS(2), and ALIAS(3) return them. If the ALIAS function cannot find a defined alias, it returns the Unknown value (?). For example, building on the previous example of three defined aliases, the functions ALIAS(4), ALIAS(5), and ALIAS(6) return the Unknown value (?) because they cannot find a defined alias.

Example

This procedure displays the aliases and logical names of all connected databases:

r-aliasf.p

```
DEF VAR i AS INT.
REPEAT i = 1 TO NUM-ALIASES:
  DISPLAY ALIAS(i) LABEL "Alias"
    LDBNAME(ALIAS(i)) LABEL "Logical Database".
END.
```

See also

CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-ALIASES function, NUM-DBS function, PDBNAME function, SDBNAME function

AMBIGUOUS function

Returns a TRUE value if the last FIND statement for a particular record found more than one record that met the specified index criteria.

Syntax

```
AMBIGUOUS record
```

record

The name of a record or record buffer used in a previous FIND statement.

To access a record in a file defined for multiple databases, you might have to qualify the record's filename with the database name. See the Record phrase reference entry for more information.

Example

The following example retrieves a customer record based on a name (cname) supplied by the user. If the procedure finds a record, it displays fields from that record. If it does not find a record because more than one record matched the selection criteria (name = cname), it displays the message: "There is more than one customer with that name." If it does not find a record because no records matched the selection criteria, it displays "Cannot find customer with that name."

r-ambig.p

```
DEFINE VARIABLE cname LIKE customer.name LABEL "Cust Name".

REPEAT:
    SET cname.
    FIND customer WHERE name = cname NO
ERROR.
    IF AVAILABLE customer
    THEN DISPLAY cust-num address city state postal-code.
    ELSE IF AMBIGUOUS customer
    THEN MESSAGE "There is more than one customer with that name".
    ELSE MESSAGE "Cannot find customer with that name".
END.
```

Sometimes the AMBIGUOUS function returns a TRUE value when there is no ambiguity. For example, if there is exactly one customer record, the following statement finds that record. Otherwise, the following statement always returns a message of “not found” rather than “ambiguous”:

```
FIND customer WHERE name BEGINS "".
```

Additionally, the following statement succeeds if there is only one Smith listed in the database:

```
FIND employee WHERE last-name = "Smith"
  AND first-name BEGINS "".
```

Note AMBIGUOUS is useful only when there is an index. If you use the AMBIGUOUS function to test a work file record, the function returns a value of FALSE because work files do not have indexes.

See also AVAILABLE function, FIND statement, LOCKED function, NEW function

AND operator

Returns a TRUE value if each logical expression is TRUE.

Syntax

```
expression AND expression
```

expression

An expression that evaluates to a logical value (TRUE or FALSE).

Example

This procedure lists all customers with credit limits between two values (supplied by the user and stored in the variables low-credit and hi-credit). The expressions credit-limit >= low-credit and credit-limit <= hi-credit are logical expressions because each yields a true or false value. Using the AND operator to join these logical expressions results in a logical expression that follows the WHERE keyword.

r-and.p

```
DEFINE VARIABLE low-credit LIKE credit-limit LABEL "Low Credit Limit".
DEFINE VARIABLE hi-credit LIKE credit-limit LABEL "High Credit Limit".

REPEAT:
  SET low-credit hi-credit WITH FRAME cr-range.
  FOR EACH customer WHERE
    (credit-limit >= low-credit) AND (credit-limit <= hi-credit):
      DISPLAY cust-num name credit-limit.
  END.
END.
```

See also

NOT operator, OR operator

APPLY statement

Applies an event to a widget or procedure.

Syntax

```
APPLY event [ TO widget-phrase ]
```

event

An expression whose value is the key code or event name that you want to apply. A special value of *event* is the value of the LASTKEY function. The LASTKEY function returns the keycode for the last event read from the user (that is, from the keyboard or mouse) or the last character read from an input file. The value *event* can be either a character-string value (event name) or an integer (key code) expression. For more information on default system actions and events, see the Events Reference.

TO *widget-phrase*

Specifies a widget or procedure to which the event is applied.

Example

This procedure shows how to use the APPLY statement to create keyboard accelerators. When you run this procedure you can invoke the trigger block attached to the order-but button by choosing the button directly or by pressing **F10** in the Name field. When you press **F10**, Progress sends the CHOOSE event to the button. This is equivalent to choosing the button with the mouse.

r-apply.p

```
DEFINE BUTTON order-but LABEL "Order"
   TRIGGERS:
      ON CHOOSE
         DO:
            FIND FIRST Order OF Customer NO-ERROR.
            IF AVAILABLE(Order)
            THEN UPDATE Order WITH FRAME upd-dlg
                     VIEW-AS DIALOG-BOX TITLE "Update Order" SIDE-LABELS.
         END.
   END TRIGGERS.

FORM
   order-but Customer.Name WITH FRAME x.

ON F10 OF Customer.Name
   DO:
      APPLY "CHOOSE" TO order-but IN FRAME x.
   END.

FIND FIRST Customer.
DISPLAY order-but Customer.Name WITH FRAME x.
ENABLE ALL WITH FRAME x.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- You can apply any event to any widget, including an insensitive widget. Most event-widget pairs have a default system action, but a few do. For example, the default system action for the A event on a fill-in widget is to insert the letter A into the fill-in at the current cursor location; however, there is no default system action for the A event on a button widget. Also, if you APPLY an event to a button, for example, the image of the button does not “depress” and then “pop back out.” Depending on the event-widget pair, the APPLY statement may or may not perform the default system action.

  Regardless of whether there is a default system action associated with an event-widget pair, you can write a trigger for the pair. The APPLY statement executes a trigger associated with an event-widget pair. If the event-widget pair has a default system action, that action occurs before or after the trigger executes, depending on the event. For more information on default system actions and events, see the next bullet in this section, and the “Events Reference” section on page 2171.

- When, in a graphical interface, you APPLY an event to a widget, you cannot easily invoke the widget animation code that runs when the user interacts with the widget physically. For example, if you APPLY a "choose" event to a button widget, you cannot easily make the image of the button move down and up, as occurs when the user clicks on the button. The difficulty exists because Progress does not provide access to the widget animation code, which resides in the windowing system. When the user clicks on the button, the windowing system detects the event, invokes the button animation code, perhaps performs other tasks, and passes the event to Progress, which invokes the trigger code associated with the event. When you APPLY a "choose" event to the button, Progress merely invokes the trigger code associated with the event. In neither case does Progress access, or provide access to, the button animation code.

  One widget that does not have this difficulty is the fill-in. When you APPLY a character-string event to a fill-in, the character string appears in the image of the fill-in. Progress accomplishes this by placing a copy of the character string into a buffer that maps to the same portion of the screen as the image of the fill-in.

- The APPLY statement serves as an important communications mechanism between procedures in an application. By defining triggers for events in a procedure, you can encapsulate functionality in the procedure. The APPLY statement allows you to access that encapsulated functionality from another procedure through a simple event interface.

- The APPLY statement is double-byte enabled. A character-string value specified for the *event* argument can contain double-byte characters.

- If a procedure calls another procedure from within an EDITING phrase and the called procedure uses the APPLY statement, the effect is the same as if the APPLY statement occurred directly within the EDITING phrase.

- If you are using APPLY in an EDITING phrase and *expression* is a key that causes a GO action (**GO**, or any key in a list used with the GO-ON option), Progress does not immediately exit the EDITING phrase but instead processes all the remaining statements in the phrase. If RETRY, NEXT, UNDO RETRY, or UNDO NEXT is executed before the end of the phrase, Progress ignores the GO and continues processing the EDITING phrase.

- APPLY -2 is the same as APPLY ENDKEY.

- For SpeedScript, you can apply an event to a procedure only.

See also ON statement, Widget phrase

ASC function

Converts a character expression representing a single character into the corresponding ASCII (or internal code page) integer value.

Syntax

```
ASC ( expression
      [ , target-codepage [ , source-codepage ] ] )
```

expression

An expression with a value of a single character that you want to convert to an ASCII (or internal code page) integer value. If *expression* is a constant, you must enclose it in quotation marks (" "). If the value of *expression* is other than a single character, ASC returns the value -1.

The values for *expression* are case sensitive. For example, ASC("a") returns a different value than ASC("A").

target-codepage

A character-string expression that evaluates to the name of a code page. The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file (a binary file that contains all of the tables that Progress uses for character management). If you supply a non-valid name, the ASC function returns the value -1 and returns a runtime error. Before returning an integer value, the ASC function converts *expression* from *source-codepage* to *target-codepage*. The returned integer value is relative to *target-codepage*. If you do not specify *target-codepage*, the value returned is the code page identified with the Internal Code Page (`-cpinternal`) parameter.

source-codepage

A character-string expression that evaluates to the name of a code page. The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file. If you supply a non-valid name, the ASC function returns the value -1. The *source-codepage* specifies the name of the code page to which *expression* is relative. The default value of *source-codepage* is the code page identified with the Internal Code Page (-cpinternal) parameter.

Example

The following procedure counts how many customers names begin with each of the letters, A-Z. It counts all other customers separately. The procedure uses the ASC function to translate a letter into an integer that it uses as an array subscript for counting.

r-asc.p

```
DEFINE VARIABLE ltrl AS INTEGER EXTENT 27.
DEFINE VARABLE i AS INTEGER.
DEFINE VARIABLE j AS INTEGER.FOR EACH customer:
  i = ASC(SUBSTRING(name,1,1)).
  IF i < ASC("A") or i > ASC("Z") THEN i = 27.
  ELSE i = i - ASC("A") + 1.
  ltrl[i] = ltrl[i] + 1.
END.DO j = 1 TO 27 WITH NO-LABELS USE-TEXT:
  IF j <= 26
  THEN DISPLAY CHR(ASC("A") + j - 1) @ ltr-name
    AS CHARACTER FORMAT "x(5)".
  ELSE DISPLAY "Other" @ ltr-name.
  DISPLAY ltrl[j].
END.
```

Notes

- The ASC function returns the corresponding value in the specified character set. By default, the value of SESSION:CHARSET is iso8859-1. You can set a different internal code page by specifying the Internal Code Page (-cpinternal) parameter. For more information, see *OpenEdge Development: Internationalizing Applications*.
- The ASC function is double-byte enabled. If the *expression* argument yields a double-byte character, this function returns a value greater than 255 and less than 65535.

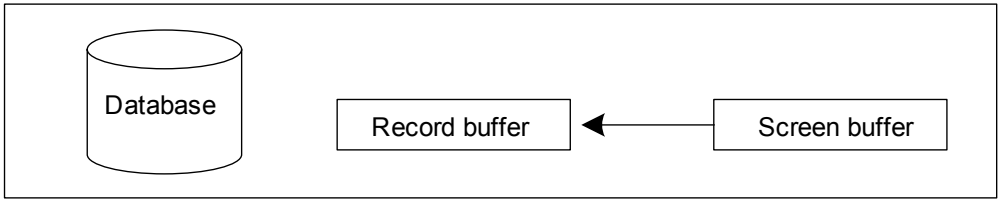
See also

CHR function, CODEPAGE-CONVERT function, INTEGER function, SESSION system handle

ASSIGN statement

Moves data previously placed in the screen buffer by a data input statement or moves data specified within the ASSIGN statement by an expression to the corresponding fields and variables in the record buffer.

Data movement



Syntax

```
ASSIGN {
         [ [ INPUT ] FRAME frame | BROWSE browse ]
           { field [ = expression ] } [ WHEN expression ]
       } ... [NO-ERROR]
```

```
ASSIGN { record [ EXCEPT field ... ] } [ NO-ERROR ]
```

```
[ FRAME frame | BROWSE browse ] field
```

The name of the field or variable (*field)* to be set from the corresponding value found in the screen buffer or expression. The *field* must be qualified by a frame name or browse name (*frame*) if *field* is specified as an input widget in more than one frame.

expression

An expression that results in an assigned value to the named field. In this case, Progress determines the *field* value from the expression rather than from the screen buffer.

WHEN *expression*

Moves data to the record buffer only when *expression* has a value of TRUE. Here, *expression* is a field name, variable name, or expression whose value is logical. Progress evaluates WHEN expressions at the beginning of the assignment, before any assignments take place.

NO-ERROR

Suppresses any errors that occur as a result of the operation. After the ASSIGN statement completes, you can check the ERROR-STATUS system handle for information about any errors that might have occurred.

If you do not specify this option and an error occurs, the assignment is canceled and any changes to field values within the assignment are undone. If the assignment occurs within a transaction, any changes to variables, work table fields, and temporary table fields are also undone, unless you define the variable or field with the NO-UNDO option.

record

The record buffer name with the fields set, from the corresponding values in the screen buffer. Naming a record is a shorthand way to list each field in that record individually.

To use ASSIGN with a record in a file defined for multiple databases, you might have to qualify the record's filename with the database name. See the Record phrase reference entry for more information.

EXCEPT *field*

All fields in the record buffer are affected except for those listed. Separate field names with a space.

Examples

The following procedure prompts you for a customer number and retrieves the customer record if one exists, or creates a new one if it does not exist. If it creates a new record, the value for the cust-num field is ASSIGNed from the value you entered in response to the PROMPT–FOR statement.

r-asgn.p

```
REPEAT:
  PROMPT-FOR customer.cust-num.
  FIND customer USING cust-num NO-ERROR.
  IF NOT AVAILABLE customer
  THEN DO:
    CREATE customer.
    ASSIGN cust-num.
  END.
  UPDATE customer WITH 2 COLUMNS.
END.
```

The next procedure changes the order number and line number of an order-line record. (It copies an order-line from one order to another.) It sets the new values into variables and modifies the record with a single ASSIGN statement that contains two assignment phrases in the form *field = expression*. Thus, both fields are changed within a single statement. Because Progress re-indexes records at the end of any statement that changes an index field value, and because `order-num` and `line-num` are used jointly in one index, this technique does not generate an index until both values change.

r-asgn2.p

```
DEFINE VARIABLE neword LIKE order-line.order-num
    LABEL "New Order".
DEFINE VARIABLE newordli LIKE order-line.line-num
    LABEL "New Order Line".
REPEAT:
  PROMPT-FOR order-line.order-num line-num.
  FIND order-line USING order-line.order-num AND line-num.
  SET neword newordli.
  FIND order WHERE order.order-num = neword.
  ASSIGN order-line.order-num = neword
    order-line.line-num = newordli.
END.
```

Notes

- If any *field* is a field in a database record, the ASSIGN statement upgrades the record lock condition to EXCLUSIVE–LOCK before updating the record.
- If any *field* is part of a record retrieved with a field list, the ASSIGN statement rereads the complete record before updating it.
- During data entry, a validation expression defined for the field in the database or in a Format phrase executes only if the widget associated with the field receives input focus. Use the VALIDATE( ) method to execute a validation expression defined for a field regardless of whether it receives input focus or not.
- Use an ASSIGN statement after a PROMPT–FOR statement or to write changes from an enabled field to the database. ASSIGN moves the value from the screen buffer into the field or variable.
- Use the PROMPT–FOR statement to receive one or more index fields from the user, and you use the FIND statement to find a record matching those index values. If no record is found, use the CREATE statement to create a new record and use the ASSIGN statement to assign the values the user supplied to the new record.
- You cannot use the SET statement in place of the PROMPT–FOR statement. The SET statement prompts the user for input and then assigns that input to the record in the buffer. However, if there is not a record available, SET cannot assign the values.
- ASSIGN does not move data into a field or variable if there is no data in the corresponding screen field. There is data in a screen field if a DISPLAY of the field was done or if data was entered into the field. If you PROMPT–FOR a field or variable that has not been DISPLAYed in the frame and enter blanks, Progress does not change the field or variable because it considers the screen field changed only if the data differs from what was in the field.
- If an ASSIGN statement references a field or variable that is used in more than one frame, it uses the value in the frame most recently introduced in the procedure.
- If you type blanks into a field that has never displayed data, the ENTERED function returns FALSE and the SET or ASSIGN statement does not update the underlying field or variable. Also, if Progress marks a field as entered, and the PROMPT–FOR statement prompts for the field again and you do not enter any data, Progress no longer considers the field entered.

- If you use a single, qualified identifier with the ASSIGN statement, the Compiler interprets the reference as *dbname.filename*. If the Compiler cannot resolve the reference as *dbname.filename*, it tries to resolve it as *filename.fieldname*.

- Many assignments within a single ASSIGN statement are more efficient than multiple ASSIGN statements. It saves r-code size and improves performance.

- The ASSIGN statement, when used in database fields, causes all related database ASSIGN triggers to execute in the order in which the fields were assigned. The ASSIGN triggers execute after all the assignments have taken place. If an ASSIGN trigger fails (or executes a RETURN statement with the ERROR option), all of the database changes are undone. See *OpenEdge Development: Progress 4GL Handbook* for more information on database triggers.

- You can assign DATE, DATETIME, and DATETIME-TZ data. When the data type expression on the left side of the assignment statement contains more information than the data type expression on the right side provides (for example, *datetime-tz = date* where a DATETIME-TZ value contains more information than a DATE value), the time value defaults to midnight and the time zone value defaults to the session's time zone. When the data type expression on the left side of the assignment statement contains less information than the data type expression on the right side provides (for example, *date = datetime-tz* where a DATE value contains less information than a DATETIME-TZ value), Progress converts the DATETIME-TZ value to the local date and time of the session, then drops the time and time zone.

- You can assign large object data from one BLOB or MEMPTR to another, and from one CLOB, LONGCHAR, or CHARACTER to another. You cannot assign large object data between BLOBs and CLOBs or MEMPTRs and LONGCHARs. You can accomplish, indirectly, by using the COPY-LOB statement. For more information, see the COPY-LOB statement reference entry.

Table 9 lists the default character conversions Progress performs when assigning CLOB, LONGCHAR, and CHARACTER data. References to CLOBCP and CLOBDB represent CLOB data in either the CLOB's defined code page or the database's defined code page, respectively. References to the "fixed code page" represent the code page of a target LONGCHAR variable set using the FIX-CODEPAGE function.

Table 9: Default character conversions with the ASSIGN statement

When the target field is a ...	And the source expression results in a ...	Progress converts the result of the source expression to ...
LONGCHAR	CLOBDB	`-cpinternal` or the fixed code page.
LONGCHAR	CLOBCP	The CLOB's defined code page or the fixed code page.
LONGCHAR	CHARACTER	`-cpinternal` or the fixed code page.
CLOBDB	CHARACTER	The database's defined code page.
CLOBDB	LONGCHAR	The database's defined code page.
CLOBCP	CHARACTER	The CLOB's defined code page.
CLOBCP	LONGCHAR	The CLOB's defined code page.
CHARACTER	CLOBDB	`-cpinternal` code page.
CHARACTER	CLOBCP	`-cpinternal` code page.
CHARACTER	LONGCHAR	`-cpinternal` code page.

- When you assign the Unknown value (?) to a BLOB or CLOB field, Progress deletes any associated object data.

See also = Assignment operator, COPY-LOB statement, INPUT function, PROMPT-FOR statement, SET statement, UPDATE statement

AT phrase

The AT phrase of the Format phrase allows explicit positioning of frame objects, either by row and column or by pixels. The AT phrase of the Frame phrase allows explicit positioning of frames with windows or parent frames.

Syntax

```
AT { COLUMN column | COLUMN-OF reference-point }
   { ROW row | ROW-OF reference-point }
   [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
```

```
AT { X x | X-OF reference-point }
   { Y y | Y-OF reference-point }
   [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
```

```
AT n
```

n

The column, measured in character units. This option is **not** supported for the Frame phrase. You cannot use the alignment options with this syntax. If you use this option, Progress chooses the row based on the previous widget and form item layout of the frame. For information on form items, see the DEFINE FRAME statement or FORM statement.

COLUMN *column*

The column, measured in character units.

COLUMN-OF *reference-point*

Indicates the column position of the field relative to another field-level widget previously defined in the frame. This option is **not** supported for the Frame phrase. This is the syntax for *reference-point*:

```
widget [ { + | - } offset ]
```

In this syntax, *widget* is a reference to a field-level widget previously defined in the frame, and *offset* is a positive decimal value. For example, if *widget* is positioned at COLUMN 10, then COLUMN-OF widget + 2.5 positions the field at column 12.5.

X *x*

The X pixel coordinate.

X-OF *reference-point*

Indicates the X co-ordinate of the field relative to another field-level widget previously defined in the frame. This option is **not** supported for the Frame phrase. The co-ordinate is expressed as the co-ordinate of a widget previously defined in the frame, plus or minus an offset. The offset must be either a constant or preprocessor constant and must be a positive integer.

ROW *row*

The row, measured in character units.

ROW-OF *reference-point*

Indicates the row of the field relative to another field-level widget previously defined in the frame. This option is **not** supported for the Frame phrase. The row is expressed as the row of a widget previously defined in the frame, plus or minus an offset. The offset must be either a constant or preprocessor constant and must be a positive decimal value.

Y *y*

The Y pixel coordinate.

`Y-OF` *reference-point*

Indicates the Y co-ordinate of the field relative to another field-level widget previously defined in the frame. This option is **not** supported for the Frame phrase. The co-ordinate is expressed as the co-ordinate of a widget previously defined in the frame, plus or minus an offset. The offset must be either a constant or preprocessor constant and must be a positive integer.

`COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED`

Specifies whether to align the left edge of the field, right edge of the field, or the colon of the field label, with the specified position. This option can only be used in combination with the ROW and COLUMN options. This option is **not** supported for the Frame phrase.

Examples

The following example uses the AT phrase to position fields within a frame:

r-at.p

```
DEFINE FRAME order-info
  order.cust-num   AT ROW 2 COLUMN 8
  customer.name    AT ROW 2 COLUMN 18
  order.order-num  AT ROW 2 COLUMN 50
  order.order-date AT ROW 2 COLUMN 65
  WITH TITLE "Order Information".

FOR EACH order NO-LOCK BREAK BY order.cust-num WITH FRAME order-info:
  IF FIRST-OF(order.cust-num) THEN
  DO:
    FIND customer OF order NO-LOCK.
    DISPLAY order.cust-num customer.name.
  END.
  DISPLAY order.order-num order.order-date.

END.
```

The following example uses relative positioning to position fields relative to the cust-num field:

r-at1.p

```
DEFINE FRAME order-info
  order.cust-num   AT X 50 Y 14
  customer.name    AT X-OF order.cust-num + 100 Y 14
  order.order-num  AT X-OF order.cust-num + 225 Y 14
  order.order-date AT X-OF order.cust-num + 320 Y 14
  WITH TITLE "Order Information" NO-LABELS.

FOR EACH order NO-LOCK
  BREAK BY order.cust-num WITH FRAME order-info:
  IF FIRST-OF(order.cust-num) THEN
  DO:
    FIND customer OF order NO-LOCK.
    DISPLAY order.cust-num customer.name.
  END.
  DISPLAY order.order-num order.order-date.
END.
```

Notes

- The AT phrase does not left justify the data. It simply specifies the position of the data area. If the data is right justified it may appear to be farther right than you expect.
- If you position a child frame completely outside the virtual area of its parent frame, Progress raises ERROR at run time when the frame is realized.
- For SpeedScript, you can position objects by row or column, not by pixels.

See also DEFINE FRAME statement, FORM statement, Frame phrase

AUDIT-ENABLED function

Determines whether a connected database is audit-enabled.

For information about audit-enabling a database, or creating and activating an audit policy for a database, see *OpenEdge Getting Started: Core Business Services*.

Syntax

```
AUDIT-ENABLED( [ integer-expression | logical-name | alias ] )
```

integer-expression

The sequence number of a connected database to query. For example, AUDIT-ENABLED(1) queries the first database, AUDIT-ENABLED(2) queries the second database, and so on. If you specify a sequence number that does not correspond to a connected database, Progress returns the Unknown value (?).

logical-name or alias

The logical name or alias of a connected database to query. These forms require a quoted character string or a character expression. If you specify a logical name or alias that does not correspond to a connected database, Progress returns the Unknown value (?).

If you specify a connected database, Progress queries that database and returns TRUE if it is audit-enabled. If you do not specify a database, Progress queries all connected databases and returns TRUE if any one of the connected databases is audit-enabled.

You can reference the AUDIT-ENABLED function within a preprocessor &IF expression (such as, &IF AUDIT-ENABLED … &ENDIF). For more information, see the &IF, &THEN, &ELSEIF, &ELSE, and &ENDIF preprocessor directives reference entry.

See also

AUDIT-POLICY system handle

AVAILABLE function

Returns a TRUE value if the record buffer you name contains a record and returns a FALSE value if the record buffer is empty.

When you use the FIND statement or the FOR EACH statement to find a record, Progress reads that record from the database into a record buffer. This record buffer has the same name as the file used by the FIND or FOR EACH statement, unless you specify otherwise. The CREATE statement creates a new record in a record buffer.

Syntax

```
AVAILABLE record
```

record

The name of the record buffer you want to check.

Note: To access a record in a file defined for multiple databases, you might have to qualify the record's filename with the database name. See the Record phrase reference entry for more information.

Example

In this procedure, the FIND statement with the NO-ERROR option bypasses the default error checking and does not display the message you get. Because item-num is unique, you do not have to use the AMBIGUOUS function to pinpoint the cause of a record not being AVAILABLE.

r-avail.p

```
REPEAT:
  PROMPT-FOR item.item-num.
  FIND item USING item-num NO-ERROR.
  IF AVAILABLE item
  THEN DISPLAY item-name price.
  ELSE MESSAGE "Not found".
END.
```

See also

AMBIGUOUS function, FIND statement, FOR statement, LOCKED function, NEW function

BASE64-DECODE function

Converts a Base64 character string into a binary value. The result is a MEMPTR containing the binary data.

Syntax

```
BASE64-DECODE ( expression )
```

expression

A CHARACTER or LONGCHAR expression containing the string you want to convert.

Example

Following is an example using the BASE64-DECODE function:

```
DEF VAR decdmptr AS MEMPTR.
DEF VAR decdlngc AS LONGCHAR.

COPY-LOB FROM FILE "C:\myicons\testencode" TO decdlngc.
decdmptr = BASE64-DECODE(decdlngc).
COPY-LOB FROM decdmptr TO FILE "C:\myicons\test.ico".
```

BASE64-ENCODE function

Converts binary data into a Base64 character string, and returns a LONGCHAR containing the character data. The resulting LONGCHAR is in the code page specified by `-cpinternal`.

Syntax

```
BASE64-ENCODE ( expression )
```

expression

A MEMPTR or RAW expression containing the binary data you want to convert.

Example

Following is an example using the BASE64-ENCODE function:

```
DEF VAR encdmptr AS MEMPTR.
DEF VAR encdlngc AS LONGCHAR.

COPY-LOB FROM FILE "C:\myicons\test.ico" TO encdmptr.
encdlngc = BASE64-ENCODE(encdmptr).
COPY-LOB FROM encdlngc TO FILE "C:\myicons\testencode".
```

BEGINS operator

Tests a character expression to see if that expression begins with a second character expression.

Syntax

```
expression1 BEGINS expression2
```

expression1

An expression that has a CHARACTER or LONGCHAR value that you test to see if it begins with *expression2*.

expression2

An expression that has a character value that you want to compare to the beginning of *expression1*. If you specify a null value ("") for *expression2*, Progress returns all the records in the database.

Examples

In this procedure, the user supplies a customer name or the first characters of a customer name. The procedure finds customer records where the name field begins with the user's input. If the customer file is indexed on the name field, this procedure is very efficient and retrieves only the selected records.

r-bgns.p

```
DEFINE VARIABLE cname LIKE customer.name LABEL "Name".

REPEAT:
  SET cname WITH SIDE-LABELS.
  FOR EACH customer WHERE name BEGINS cname:
    DISPLAY name address city state postal-code.
  END.
END.
```

The next procedure lists exactly the same customers. However, it is much less efficient because it retrieves and examines all customer records, and only displays the ones with the appropriate names.

r-bgns2.p

```
DEFINE VARIABLE cname LIKE customer.name LABEL "Name".

REPEAT:
    SET cname WITH SIDE-LABELS.
    /* create MATCHES pattern */
    cname = cname + "*".
    FOR EACH customer WHERE name MATCHES cname:
        DISPLAY name address city state postal-code.
    END.
END.
```

Notes

- The BEGINS operator is double-byte enabled. You can use the BEGINS operator to compare strings containing double-byte characters.
- When you use the BEGINS operator to compare LONGCHAR fields, BEGINS always uses the `-cpcoll` collation.
- BEGINS is useful in a WHERE phrase that specifies which records should be retrieved in a FOR EACH block. Unlike the MATCHES operator, which requires that all records in the file be scanned, BEGINS uses an index wherever possible.
- Most character comparisons are case insensitive in Progress. By default, all characters are converted to uppercase prior to comparisons. However, you can define fields and variables as case sensitive (use if strict ANSI SQL adherence is required). If **either** of the character expressions passed to BEGINS is a field or variable defined as case sensitive, the comparison is case sensitive. In a case-sensitive comparison “SMITH” does not equal “Smith”.

- Progress considers trailing blanks in the BEGINS operator. For example, this statement is FALSE:

```
"x" BEGINS "x      "
```

This is different than comparisons, where trailing blanks are ignored. For example, this statement IS TRUE:

```
"x" = "x      "
```

See also MATCHES operator

BELL statement

Causes the terminal to make a beep sound.

Note: Does not apply to SpeedScript programming.

Syntax

```
BELL
```

Example

The following procedure dynamically determines the output file to use for a report that lists all customer records. The SET statement gets the name of a file from the user. The SEARCH function returns an unqualified file name if that file already exists in your working directory. If the file exists in your working directory, it displays messages, undoes the work done in the DO block, and lets the user enter another file name. (The procedure determines whether the file is in your working directory. If SEARCH returns a directory other than your current working directory, you receive no messages and it does not undo your work.) After you type a file name that does not already exist, the OUTPUT TO statement directs the output of the procedure to that file.

r-bell.p

```
DEFINE VARIABLE outfile AS CHARACTER FORMAT "x(8)"
    LABEL "Output file name".

getfile:
DO ON ERROR UNDO, RETRY:
    SET outfile WITH SIDE-LABELS.
    IF SEARCH(outfile) = outfile THEN DO:
       MESSAGE "A file named" outfile "already exists in your directory".
       MESSAGE "Please use another name".
       BELL.
       UNDO getfile, RETRY getfile.
    END.
END.

OUTPUT TO VALUE(outfile).

FOR EACH customer:
    DISPLAY name credit-limit.
END.
```

Note

If the terminal is not the current output device, BELL has no effect.

BUFFER-COMPARE statement

Performs a bulk comparison of two records (source and target) by comparing source and target fields of the same name for equality and storing the result in a field. You can specify a list of fields to exclude, or a list of fields to include. You can also specify WHEN...THEN phrases. For all such phrases you specify, Progress evaluates the WHEN portion, and if it evaluates to TRUE, Progress executes the THEN portion.

Syntax

```
BUFFER-COMPARE source
  [ { EXCEPT | USING } field ... ] TO target
  [ CASE-SENSITIVE | BINARY ]
  [ SAVE [ RESULT IN ] result-field ]
  [ [ EXPLICIT ] COMPARES ]:
     [ WHEN field compare-operator expression
        THEN statement-or-block ] ...
  [ END [ COMPARES ] ] [ NO-LOBS ] [ NO-ERROR ]
```

source

The source database table, buffer, temporary table, or work table.

EXCEPT *field*

A list of source fields to exclude from the bulk compare.

USING *field*

A list of source fields to include in the bulk compare. The USING option is a positive version of the EXCEPT option.

TO *target*

The target database table, buffer, temporary table, or work table.

CASE-SENSITIVE

Directs Progress to perform a case-sensitive comparison.

BINARY

Directs Progress to perform a binary comparison.

SAVE RESULT IN *result-field*

A variable or field to contain the result of the comparison. The variable or field must be CHARACTER or LOGICAL.

If *result-field* is CHARACTER, the result is a comma-separated list of fields that failed the comparison, sorted in ascending order.

If *result-field* is LOGICAL, the result is YES if all fields are equal, or NO if any fields are unequal. In either case, BUFFER-COMPARE stops comparing when it encounters the first inequality.

EXPLICIT COMPARES

Opens a block of WHEN options. If you open the block, you must close it with END COMPARES.

WHEN *field*

Any data field in the source.

BUFFER-COMPARE removes this field from a USING list or adds this field to an EXCEPT list. This removes the field from the bulk compare and from result-field.

compare-operator

Represents one of the following: LT, LE, GT, GE, EQ, NE, MATCHES, BEGINS, or CONTAINS.

expression

Any valid Progress expression.

THEN *statement-or-block*

Any Progress statement or block. The statement or block executes when the WHEN clause evaluates to TRUE.

`END COMPARES`

Closes the block of WHEN phrases.

`NO-LOBS`

Directs Progress to ignore large object data when comparing records that contain BLOB or CLOB fields.

`NO-ERROR`

Diverts any error messages from this statement to the ERROR-STATUS system handle and records the success of this statement in ERROR-STATUS:ERROR.

Notes

- At compile time, BUFFER-COMPARE:
  - Fails to compile if any source-target field pair is not type compatible. An example of such a pair is a field that is LOGICAL in the source, but DECIMAL in the target.
  - Excludes from the bulk comparison all EXCEPT *field* fields and all WHEN *field* fields.
  - Automatically excludes from the bulk comparison fields that appear in the source but not in the target.
  - Tries to bind unqualified field names that appear in the EXCEPT and USING options to the source buffer.
- At run time, BUFFER-COMPARE:
  - Compares all fields not in the EXCEPT phrase and all fields not in the WHEN phrase for equality.
  - Stores the result in the field that the SAVE phrase specifies, if any.
  - Evaluates each WHEN option, executing it if its condition evaluates to TRUE.

Note: This behavior is different from the behavior of the Progress 4GL CASE statement, which executes only the first WHEN option whose condition evaluates to TRUE.

- When comparing records that contain BLOB fields, Progress performs a binary comparison on the BLOB data associated with the source and target records, and reports the results of the comparison.
- You cannot use the BUFFER-COMPARE statement to compare records that contain CLOB fields, unless one or both of the corresponding fields contain the Unknown value (`?`). However, you can convert CLOB fields to LONGCHAR values and use the EQ, GE, GT, LE, LT, or NE comparison operator to compare the LONGCHAR values.
- Use the `NO-LOBS` option with the BUFFER-COMPARE statement to ignore large object data when comparing records that contain BLOB or CLOB fields. You can also use the `EXCEPT` option to exclude BLOB and CLOB fields from the compare.

See also BUFFER-COPY statement

BUFFER-COPY statement

Performs a bulk copy of a source record to a target record by copying each source field to the target field of the same name. You can specify a list of fields to exclude from the bulk copy, or a list of fields to include in the bulk copy. You can also specify WHEN...THEN phrases. For each such phrase, BUFFER-COPY executes the THEN portion if the corresponding WHEN portion evaluates to TRUE.

Syntax

```
BUFFER-COPY source [ { EXCEPT | USING } field ... ]
  TO target [ ASSIGN assign-expression ... ] [ NO-LOBS ] [ NO-ERROR ]
```

source

The source database table, buffer, temporary table, or work table.

EXCEPT *field* **...**

A list of space-separated source fields to exclude from the bulk copy.

USING *field* **...**

A list of space-separated source fields to include in the bulk copy. The USING option is simply a positive version of the EXCEPT option.

TO *target*

The source database table, buffer, temporary table, or work table.

ASSIGN *assign-expression*

A space-separated list of any valid Progress 4GL ASSIGN statements (without the EXCEPT option, which BUFFER-COPY already provides). BUFFER-COPY performs each *assign-expression* and automatically excludes the field on the left side ("destination") of each *assign-expression* from the bulk copy-except for field extents (subscripted fields). If a field extent appears on the left side of an assign-expression, BUFFER-COPY does not automatically exclude that extent (such as customer.mnth-sales[1]) or the field as a whole (such as customer.mnth-sales) from the bulk copy.

NO-LOBS

Directs Progress to ignore large object data when copying records that contain BLOB or CLOB fields.

NO-ERROR

Diverts any error messages from this statement to the ERROR-STATUS system handle and records the success of this statement in ERROR-STATUS:ERROR.

Notes

- At compile time, BUFFER-COPY:
  - Fails to compile if any source-target field pair is not type compatible.
  - Excludes from the bulk copy all EXCEPT *field* fields, and all *assign-expression* fields on the left side of the assignment.
  - Automatically excludes fields that appear in the source but not the target from the bulk copy.
  - Tries to bind unqualified field names that appear in the EXCEPT and USING options to the source buffer.
- At run time, BUFFER-COPY:
  - Creates a target record if none already exists and executes any applicable CREATE triggers.
  - Assigns all matching fields that do not appear in the EXCEPT or ASSIGN options.
  - Performs each assign-expression in the ASSIGN option, one-by-one.
- The BUFFER-COPY statement, like the VALIDATE statement, must appear within the scope of a FIND, a FOR EACH, or a CREATE statement that references the source table.

- If a BUFFER-COPY statement references a target buffer for the first time, Progress regards this reference as a “free reference” and scopes the buffer to the nearest enclosing block that can scope records. For more information on free references, see the chapter on block properties in *OpenEdge Development: Progress 4GL Handbook*.
- With respect to transaction processing, Progress treats a BUFFER-COPY statement the same way it would treat equivalent ASSIGN statements. For more information on transaction processing, see the chapter on transactions in *OpenEdge Development: Progress 4GL Handbook*.
- The compiler’s XREF facility automatically creates a REFERENCE for each field in the fields list, a TABLE-REFERENCE for the source and target buffers, ACCESS and UPDATE references for any fields in the ASSIGN option, and ACCESS (or UPDATE) references for each source (or target) field that participates in the bulk copy.
- When copying records that contain a BLOB or CLOB field, Progress copies the object data associated with the source record to the target record. If the BLOB or CLOB field in the source record contains the Unknown value (?), Progress stores the Unknown value (?) in the BLOB or CLOB field of the target record. If the target record already has object data associated with it, Progress deletes that object data before copying the new object data.
- Use the `NO-LOBS` option with the BUFFER-COPY statement to ignore large object data when copying records that contain BLOB or CLOB fields. More specifically:
  - When you copy a source record to a new target record, Progress sets the value of the BLOB or CLOB field in the target record to the Unknown value (?).
  - When you copy a source record to an existing target record, Progress does not change the value of the BLOB or CLOB field in the existing target record.

  You can also use the `EXCEPT` option to exclude BLOB and CLOB fields from the copy.

See also BUFFER-COMPARE statement

CALL Statement

Transfers control to a dispatch routine (PRODSP) that then calls a C function. You write the C function using Progress Host Language Call (HLC) interface.

Progress HLC consists of a collection of C functions that:

- Obtain data from Progress shared variables and buffers
- Set data in Progress shared variables and buffers
- Control screen modes
- Provide Progress-like messages in the message area at the bottom of the screen

Using HLC, you can extend Progress with your own C functions.

Syntax

```
CALL routine-identifier [ argument ] ...
```

routine-identifier

The name the PRODSP dispatch routine used to identify the C function to call.

argument

One or more arguments that you want to pass to the C function.

See also

RUN statement

CAN-DO function

Checks a string value against two types of comma-separated lists:

- An ID list of one or more user permission strings that indicate what users have access to the current procedure. The function returns TRUE if the specified user ID has access according to the list. Thus, you can implement run-time authorization for any procedure in your application.
- An arbitrary list of string values. The function returns TRUE if the specified string value is contained in the list.

Syntax

```
CAN-DO ( id-list [ , string ] )
```

idlist

A constant, field name, variable name, or expression that evaluates to a list of one or more user IDs. If the expression contains multiple user IDs, you must separate the user IDs with commas. Do not insert blanks between the user IDs

Table 10 lists values you can use in *idlist*.

Table 10: Values to use for ID lists

Value	Meaning
*	All users have access.
user	This user has access.
!*user*	This user does not have access.
*string**	Users whose IDs begin with *string* have access.
!*string**	Users whose IDs begin with *string* do not have access.

You can use any combination of values to define *idlist*, and you must separate the values with commas.

string

A character expression. The *string* is checked against *idlist*. If you do not enter *string*, the compiler inserts the USERID function that is evaluated each time you run the procedure. If the compiler inserts the USERID function, it does not reference a database name. If you use the USERID function and have more than one database connected, be sure to include the database name, for example, USERID “demo”.

Examples

The `r-cando.p` procedure is based on an activity permission table called `permission`. The permission table is not included in your demo database. However, the records in that table might look something like the following:

Activity	Can-Run
custedit	manager,salesrep
ordedit	manager,salesrep
itemedit	manager,inventory
reports	manager,inventory,salesrep

In `r-cando.p` the FIND statement reads the record for the activity custedit in the permission table. (This assumes that a unique primary index is defined on the activity field.) The CAN-DO function compares the user ID of the user running the procedure with the list of users in the can-run field of the custedit record. If the user ID is manager or salesrep, the procedure continues executing. Otherwise, the procedure displays a message and control returns to the calling procedure.

r-cando.p

```
DO FOR permission:
  FIND permission "custedit".
  IF NOT CAN-DO(permission.can-run)
  THEN DO:
    MESSAGE "You are not authorized to run this procedure".
    RETURN.
  END.
END.
```

In this next example, the CAN-DO function compares *userid* (the user ID for the current user) against the values in *idlist*. The values in *idlist* include manager and any user IDs beginning with acctg except acctg8. If there is no match between the two values, the procedure displays a message and then exits.

r-cando2.p

```
IF NOT CAN-DO("manager,!acctg8,acctg*")
THEN DO:
  MESSAGE "You are not authorized to run this procedure.".
RETURN.
END.
```

In addition to performing security checks, you can use the CAN-DO function for looking up any value in a comma-separated list. For example, the following procedure searches your PROPATH for your DLC directory:

r-cando3.p

```
MESSAGE "The DLC directory " +
        (IF CAN-DO(PROPATH, OS-GETENV("DLC")) THEN "is" ELSE "is NOT") +
        " in your PROPATH.".
```

Notes

- If *idlist* contains contradictory values, the first occurrence of a value in the list applies. For example, CAN-DO(“abc,!abc*”,“abc”) is TRUE, since the user ID abc appears before!abc in *idlist*.
- If *idlist* is exhausted without a match, CAN-DO returns a value of FALSE. Therefore, !abc restricts abc and everyone else (including the blank userid, ""). To restrict abc only and allow everyone else, use !abc,*.
- A *userid* comparison against *idlist* is not case sensitive.
- If a user is logged into the system as root, Progress allows access to the procedure even if access is denied by the *idlist.* You must specifically deny root access by adding !root to the *idlist*.

- In addition to the examples shown above, you can use the CAN-DO function to compare a *userid* other than that of the current user against the list of values in *idlist*. For example, to assign a department *userid* to users “smith” and “jones” when they start Progress, you can prompt these users for a department *userid* and password. Progress then compares the supplied information against a table of identifiers.

  If the values supplied by the user match those in the identifier table, you can define a global shared variable for Progress to use for the entire session. The value of this variable is the department *userid*. Progress uses the CAN-DO function to compare *userid* (the value of the global shared variable) against the list of values in *idlist*.

  If you know the name of the global shared variable, you can define another variable with the same name and call subroutines directly.

- You establish user IDs with the USERID and SETUSERID functions, or with the Userid (-U) parameter and Password (-P) parameter. The user ID can be an operating system user ID (on UNIX) or a user ID stored in the Progress _User table (in Windows or on UNIX).

- Progress returns a Compiler error if you omit *userid* and one of the following conditions exists:

  – There is no database connected.

  – More than one database is currently connected.

- CAN-DO outside of a VALIDATE statement is the same as FIND ... NO-ERROR followed by IF AVAILABLE(...).

See also SETUSERID function, USERID function, VALIDATE statement

CAN-FIND function

Returns a TRUE value if a record is found that meets the specified FIND criteria; otherwise it returns FALSE. CAN-FIND does not make the record available to the procedure. You typically use the CAN-FIND function within a VALIDATE option in a data handling statement, such as the UPDATE statement.

You can use CAN-FIND to see if a record exists with less system overhead than that of a FIND statement. The query capabilities are similar. CAN-FIND is also useful for implementing inner joins among database tables.

Syntax

```
CAN-FIND
  (
    [ FIRST | LAST ] record [ constant ]
    [ OF table ] [ WHERE expression ] [ USE-INDEX index ]
    [ USING [ FRAME frame ] field
       [ AND [ FRAME frame ] field ] ...
    ]
    [ SHARE-LOCK | NO-LOCK ] [ NO-WAIT ] [ NO-PREFETCH ]
  )
```

You can specify the OF, WHERE, USE-INDEX, and USING options in any order.

FIRST

Returns TRUE if CAN-FIND locates a record that meets the specified criteria; otherwise returns FALSE.

LAST

Returns TRUE if CAN-FIND locates a record that meets the specified criteria; otherwise returns FALSE.

record

The record buffer you are checking for existence.

To use CAN-FIND to locate a record in a table defined for multiple databases, you might have to qualify the record's table name with the database name. See the Record phrase reference entry for more information.

constant

The table you want to use has a primary index; the *constant* is the value of the last component field of that index for the record you want.

OF *table*

Qualifies the records to use by relating the record to a record in another table.

WHERE *expression*

Qualifies the record that CAN-FIND searches for. The expression must return a TRUE or FALSE value.

USE-INDEX *index*

Identifies the index you want CAN-FIND to use to find a record. If you do not use this argument, Progress selects an index to use based on the criteria specified with the WHERE, USING, OF, or *constant* arguments.

USING [FRAME *frame*] *field* [AND [FRAME *frame*] *field*]

One or more names of fields you want to use to search for a record. The field you name in this argument must have been previously entered into a screen field, usually with a PROMPT-FOR statement. The field must be viewed as a fill-in or text widget.

SHARE-LOCK

Specifies that CAN-FIND determines whether the record can be SHARE-LOCKed. If you use this option without the NO-WAIT option, and if the record is EXCLUSIVE-LOCKed, CAN-FIND waits until that lock is released before returning. If you use SHARE-LOCK with the NO-WAIT option, then CAN-FIND returns a FALSE value immediately if the record is EXCLUSIVE-LOCKed.

NO-LOCK

Specifies that CAN-FIND determines whether the record can be accessed with the NO-LOCK option. This is the default for CAN-FIND.

NO-WAIT

Causes CAN-FIND to return immediately and return FALSE if the record is locked by another user.

If you use NO-WAIT together with a SHARE-LOCK and the record found is EXCLUSIVE-LOCKed, the CAN-FIND function does not wait and returns FALSE.

NO-PREFETCH

Specifies that only one record can be sent across the network at a time. If you do not specify this option, Progress might send more than one record from the server to the client in each network packet.

Example

In the following procedure, the UPDATE statement uses the VALIDATE option to make sure that the salesrep entered matches one of the salesreps in the database. The VALIDATE option uses the CAN-FIND function to find the record.

r-canfnd.p

```
REPEAT:
  CREATE customer.
  UPDATE cust-num name sales-rep
    VALIDATE(CAN-FIND(sales-rep WHERE
      salesrep.sales-rep = customer.sales-rep),
      "Invalid sales rep -- please re-enter").
END.
```

Notes

- Fields do not have to be indexed to use them in a CAN-FIND function. For example, you can use the following CAN-FIND function with the sports database, even though the state field is not indexed:

  ```
  CAN-FIND(FIRST customer where state = "NH")
  ```

  However, when you use CAN-FIND on a non-indexed field, the response might be slow, as with a FIND.

- You can name more than one field as part of the selection criteria. For example, the following CAN-FIND function works with the sports database:

```
CAN-FIND(customer WHERE cust-num = x AND name = y)
```

- CAN-FIND supports selection criteria that uses inequality matches. Therefore, you can use Boolean operations in WHERE clauses.
- EXCLUSIVE lock is not allowed in a CAN-FIND because CAN-FIND does not return a record.
- If you use the CAN-FIND function to find a record in a work table, Progress disregards the NO-WAIT, SHARE-LOCK, and NO-LOCK options.
- You can nest CAN-FIND functions. For example, you can use CAN-FIND(... WHERE CAN-FIND(...WHERE CAN-FIND, etc.
- The CAN-FIND function does not cause FIND triggers to execute; hence a procedure can use this function to bypass the FIND trigger and check for the existence of records. Anyone writing a FIND trigger for security reasons should be aware of this.
- You cannot use the CAN-FIND function in a WHERE clause. Doing so generates a compiler error.
- Within a CAN-FIND function, if you compare tables or fields from multiple databases, you must explicitly specify the database name along with the table and field name.

See also FIND statement

CAN-QUERY function

Returns a logical value indicating whether you can query a specified attribute or method for a specified widget.

Syntax

```
CAN-QUERY ( widget-handle , attribute-name )
```

widget-handle

An expression that evaluates to a widget handle. The handle must refer to a valid widget.

attribute-name

An expression that evaluates to a character-string value. The contents of the string must be an attribute or method name. For more information on attributes, see the "Attributes and Methods Reference" section on page 1497.

Example

The following example prompts for a widget type and an attribute. It creates a widget of the specified type and passes a handle to that widget and the attribute you specified to the CAN-QUERY and CAN-SET functions. Then it reports whether the attribute can be queried or set for that widget.

r-prog.p

```
DEFINE VARIABLE attribute AS CHARACTER FORMAT "x(24)" LABEL "Attribute".
DEFINE VARIABLE queryable AS LOGICAL VIEW-AS TOGGLE-BOX LABEL "Query".
DEFINE VARIABLE setable   AS LOGICAL VIEW-AS TOGGLE-BOX LABEL "Set".
DEFINE VARIABLE temp-handle AS WIDGET-HANDLE.
DEFINE VARIABLE widget-type AS CHARACTER FORMAT "x(24)" LABEL "Widget".

FORM
   widget-type attribute setable queryable.

REPEAT:
   UPDATE widget-type attribute.
   CREATE VALUE(widget-type) temp-handle.
   queryable = CAN-QUERY(temp-handle, attribute).
   setable = CAN-SET(temp-handle, attribute).
   DISPLAY queryable setable.
   DELETE WIDGET temp-handle.
END.
```

Note

For SpeedScript, use with buffer-field, buffer-object, buffer, and query-object handles.

See also

CAN-SET function, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function

CAN-SET function

Returns a logical value indicating whether you can set a specified attribute for a specified widget.

Syntax

```
CAN-SET ( widget-handle , attribute-name )
```

widget-handle

An expression that evaluates to a widget handle. The handle must refer to a valid widget.

attribute-name

An expression that evaluates to a character-string value. The contents of the string must be an attribute name. For more information on attributes, see the "Attributes and Methods Reference" section on page 1497.

Notes

- For a field-level widget, the CAN-SET function always returns TRUE for the FRAME attribute; however, you can set the frame attribute only if the widget is dynamic. Therefore, before setting the FRAME attribute for a widget, you can test that the operation is valid with a statement similar to the following:

```
IF CAN-SET(my-handle, "FRAME") AND my-handle:DYNAMIC
THEN my-handle:FRAME = frame-handle.
```

- For SpeedScript, use with buffer-field, buffer-object, buffer, and query-object handles.

See also

CAN-QUERY function, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function

CAPS function

Converts any lowercase letters in a character string expression to uppercase characters, and returns the resulting character string.

Syntax

```
CAPS ( expression )
```

expression

A constant, field name, variable name, or expression that results in a character string.

Example

In the following code example, the CAPS function converts the characters in the state field to uppercase:

r-caps.p

```
REPEAT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING cust-num.
    UPDATE name address city state.
    customer.state = CAPS(customer.state).
    DISPLAY customer.state.
END.
```

Notes

- The CAPS function returns uppercase characters relative to the settings of the Internal Code Page (-cpinternal) and Case Table (-cpcase) startup parameters. For more information on these parameters, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- The CAPS function is double-byte enabled. The specified expression can yield a string containing double-byte characters; however, the CAPS function changes only single-byte characters in the string.

See also

LC function

CASE statement

Provides a multi-branch decision based on the value of a single expression.

Syntax

```
CASE expression :
  { WHEN value [ OR WHEN value ] ... THEN
      { block | statement }
  } ...
  [ OTHERWISE
      { block | statement }
  ]
END [ CASE ]
```

expression

The expression that determines which branch of code to execute. The *expression* parameter can be any valid Progress expression. It can include comparisons, logical operations, and parentheses.

WHEN *value* [OR WHEN *value*] . . . THEN

Each *value* is an expression that evaluates to a possible value for *expression*. If *value* matches the current value of *expression*, then the associated block or statement executes.

OTHERWISE

Introduces a block or statement to execute when the value of *expression* does not match any *value* in any of the WHEN clauses.

block

A DO, FOR EACH, or REPEAT block. If you do not use a block, then you can only use a single statement for the WHEN or OTHERWISE clause.

statement

A single 4GL statement. If you want to use more than one statement, you must enclose them in a DO, FOR EACH, or REPEAT block.

END [CASE]

Indicates the end of the CASE statement. You can include the CASE keyword here to improve readability; it has no effect on the code.

Example The following fragment shows a simple example of a CASE statement:

r-case.p

```
DEFINE VARIABLE pay-stat    AS INTEGER INITIAL 1.

UPDATE pay-stat VIEW-AS RADIO-SET
  RADIO-ITEM unpaid   1 LABEL "Unpaid"
  RADIO-ITEM part     2 LABEL "Partially paid"
  RADIO-ITEM paid     3 LABEL "Paid in full".

CASE pay-stat:
  WHEN 1 THEN
    MESSAGE "This account is unpaid.".
  WHEN 2 THEN
    MESSAGE "This account is partially paid.".
  WHEN 3 THEN
    MESSAGE "This account is paid in full.".
END CASE.
```

Notes

- Each *value* must have the same data type as *expression*. If the data types do not match, the compiler reports an error.
- You can specify any number of WHEN clauses within the CASE statement.
- You can specify only one OTHERWISE clause for a CASE statement. If you use the OTHERWISE clause, it must be the last branch in the statement.
- When a CASE statement is executed, Progress evaluates *expression* and evaluates each *value* for each branch in order of occurrence until it finds the first *value* that satisfies the condition. At that point Progress executes that branch and does not evaluate any other *value* for that branch or any other branches. If no matching *value* is found, then the OTHERWISE branch is executed, if given. If the OTHERWISE branch is not given and no matching *value* is found, then no branch of the CASE statement is executed and execution continues with the statement after the CASE statement.
- After a branch of the CASE statement is executed, Progress leaves the CASE statement and execution continues with the statement following the CASE statement.
- If a LEAVE statement is executed within any branch of a CASE statement, Progress leaves the closest block (other than a DO block) that encloses the CASE statement.

CAST function

Casts an object reference of one class type to another class type within a class hierarchy.

When you cast an object reference, Progress treats the object reference as if it were an instance of the class type to which it was cast. The underlying object instance does not change.

Syntax

```
CAST( object-reference, type-name ).
```

object-reference

An object reference for the user-defined class object instance to cast.

type-name

A character string that specifies the type name of the class or interface to which the object reference is cast. This class or interface must be in the hierarchy of the defined class. Specify a type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

Notes

- You typically cast an object reference down a class hierarchy (that is, from a super class to a subclass within a class hierarchy). Since a subclass contains all the super classes in its inherited class hierarchy, Progress can implicitly cast an object reference up a class hierarchy.
- At compile time, Progress verifies that the specified class type is within the class hierarchy of the specified object reference. At run time, Progress checks the validity of the cast operation.
- You can use the CAST function to cast an object reference to a subclass and invoke a method defined in that subclass using the following syntax:

```
CAST( object-reference, type-name ):method-name( ).
```

You can also use the CAST function to cast a parameter in a parameter list for a method using the following syntax:

```
method-name( INPUT CAST( object-reference, subclass-name ), ... ).
```

You can also use the CAST function to cast a temp-table field, which is defined as a Progress.Lang.Object, to use as an object of another class type. For example:

```
DEFINE TEMP-TABLE mytt FIELD CustObj AS Progress.Lang.Object.
DEFINE VARIABLE RCustObj AS CLASS acme.myObjs.CustObj.

RCustObj = CAST(mytt.CustObj, "acme.myObjs.CustObj").
```

You can now use the object reference in `RCustObj` to invoke methods in the `acme.myObjs.CustObj` class.

CHOOSE statement

After you display data, the CHOOSE statement moves a highlight bar among a series of choices and selects a choice when you press **GO**, **RETURN**, or enter a unique combination of initial characters.

This statement is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
CHOOSE
  {     { ROW field [ HELP char-constant ] }
    |   { FIELD { field [ HELP char-constant ] } ... }
  }
  [ AUTO-RETURN ] [ COLOR color-phrase ]
  [ GO-ON ( key-label ... ) ] [ KEYS char-variable ]
  [ NO-ERROR ] [ PAUSE expression ]
  { [ frame-phrase ] }
```

You can specify the AUTO-RETURN, COLOR, GO-ON, KEYS, NO-ERROR, and PAUSE options in any order.

`ROW field`

Tells CHOOSE to move a highlight bar among iterations of a down frame. The *field* is the name of the field that you want the highlight bar to begin highlighting. The ROW option is useful for browsing through a set of records, although *field* does not have to refer to database records.

If you use the ROW option with the CHOOSE statement, Use the SCROLL statement as well. See the SCROLL statement reference entry examples.

If you use ROW, you can add a COLOR statement to control the video display highlighting.

FIELD *field*

Tells CHOOSE to move a highlight bar through a set of fields or set of array elements in a frame. The *field* argument is the table record or array variable with fields or elements through which you want to move the highlight bar. These fields or array elements must be defined as Progress default FILL-IN widgets (not specified with the FILL-IN NATIVE option). The FIELD option is useful for building menus. You can also supply help for *field*.

HELP *char-constant*

Lets you provide help text for each field in a CHOOSE FIELD statement or for the entire CHOOSE ROW statement. For the CHOOSE ROW statement, the help text is displayed throughout the CHOOSE operation. For the CHOOSE FIELD statement, the help text you specify for a field is displayed whenever you move to the field.

AUTO-RETURN

Tells Progress to use the selection when you enter a unique string of initial characters. When you use AUTO-RETURN and the user enters a unique string of initial characters, Progress sets the value of LASTKEY to KEYCODE (return).

COLOR *color-phrase*

Specifies a video attribute or color for the highlight bar. Following is the syntax for *color-phrase*:

```
{     NORMAL
    | INPUT
    | MESSAGES
    | protermcap-attribute
    | dos-hex-attribute
    | { [ BLINK- ] [ BRIGHT- ]
        [ fgnd-color ] [ bgnd-color ] }
    | { [ BLINK- ] [ RVV- ] [ UNDERLINE- ] [ BRIGHT- ]
        [ fgnd-color ] }
    | VALUE ( expression )
}
```

For more information on *color-phrase*, see the COLOR phrase reference entry.

GO-ON (*key-label*) . . .

Names *key-labels* for keys that cause CHOOSE to return control to the procedure. If you do not use the GO-ON option, CHOOSE returns control to the procedure when the user presses **GO**, **RETURN**, **END-ERROR**, or types a unique substring when AUTO-RETURN is in effect. If you don't specify **F1**, **RETURN**, or **F4**, those keys are still GO-ON keys by default.

KEYS *char-variable*

If you want to highlight a particular choice when entering a CHOOSE statement, or if you want to know what keys the user pressed to make a selection, use the KEYS option. When you use the KEYS option, you must give the name of a character variable, *char-variable*. If *char-variable* is initialized to one of the choices before entering the CHOOSE statement, Progress highlights that choice. As the user presses keys to move the highlight bar, Progress saves those keystrokes in *char-variable*. You can test the value of *char-variable* after the CHOOSE statement returns control to the procedure. There is a 40-character limit when using the KEYS option.

NO-ERROR

Overrides default error handling by the CHOOSE statement, and returns control to the procedure. If you do not use the NO-ERROR option, the CHOOSE statement causes the terminal to beep when the user presses an invalid key.

If you use the NO-ERROR option and the user presses an invalid key, the CHOOSE statement ends. At this point, you usually want to use the LASTKEY function to test the value of the last key the user pressed and then take the appropriate action.

Note that the NO-ERROR option of the CHOOSE statement does not have any affect on the ERROR-STATUS system handle.

PAUSE *expression*

Specifies a time-out period in seconds. If the user does not make a keystroke for the specified number of seconds, the CHOOSE statement times out and returns control to the procedure. The time-out period begins before the user's first keystroke and is reset after each keystroke. If CHOOSE times out, the value of LASTKEY is -1. Use time-out period to prevent inactivity.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

If your procedure might eventually run on a spacetaking terminal, use the ATTR-SPACE option for the CHOOSE statement. Omitting this option makes the highlight bar invisible.

Example

The following procedure displays a strip menu with four choices. The procedure defines two arrays; one holds the items for selection on the menu, the other holds the names of the programs associated with the menu selections. The CHOOSE statement allows the user to select an item from the strip menu. Progress finds the number (within the array) associated with the item selected and the program associated with that number in the proglist array. Progress runs the program, if it exists, and displays a message. It also allows the user to select another item if the program does not exist. (In your own application, you associate actions with items selected by the CHOOSE statement.)

r-chsmnu.p

```
DEFINE VARIABLE menu AS CHARACTER EXTENT 4 FORMAT "x(7)"
  INITIAL [ "Browse", "Create", "Update", "Exit" ].
DEFINE VARIABLE proglist AS CHARACTER EXTENT 4
  INITIAL [ "brws.p", "cre.p", "upd.p", "exit.p"].

FORM "Use the sample strip menu to select an action."
  WITH FRAME instruc CENTERED ROW 10.

REPEAT:
  VIEW FRAME instruc.
  DISPLAY menu WITH NO-LABELS ROW 21 NO-BOX ATTR-SPACE
    FRAME f-menu CENTERED.
  HIDE MESSAGE.
  CHOOSE FIELD menu GO-ON (F5) AUTO-RETURN
    WITH FRAME f-menu.
  IF SEARCH(proglist[FRAME-INDEX]) = ?
  THEN DO:
    MESSAGE "The program" proglist[FRAME-INDEX] "does not exist.".
    MESSAGE "Please make another choice.".
  END.
  ELSE RUN VALUE(proglist[FRAME-INDEX]).
END.
```

The GO-ON option sets the **GET** key to perform an action like **GO**. With the LASTKEY function, you could check for **F5** and take another action relevant to your application.

Notes

- If you do not specify help text in the CHOOSE statement, any help text you specify for the field in the Data Dictionary is displayed instead. If no help text is specified in either the CHOOSE statement or Data Dictionary, then the status default message is displayed throughout the CHOOSE statement.
- The CHOOSE statement takes different actions depending on the key you press and whether you use the NO-ERROR option, as shown in Table 11.

Table 11: CHOOSE statement actions *(1 of 2)*

Key	NO-ERROR	Action
Valid cursor motion.[1]	N/A	Clear saved keys, move highlight bar.
Invalid cursor motion.[2]	NO	Clear saved keys, beep terminal.
Invalid cursor motion.[2]	YES	Clear saved keys, return control to procedure.
A non-unique string followed by an alphanumeric character that does not form a matchable string.[3]	NO	Clear saved keys, try to match the last key entered. If no match is available, beep terminal.
A non-unique string followed by an alphanumeric character that does not form a matchable string with the other characters.	YES	Return control to procedure.
An invalid string.	NO	Beep terminal.
An invalid string.	YES	Return control to the procedure and, if the KEYS option was used, save any printable keys.

Table 11: CHOOSE statement actions *(2 of 2)*

Key	NO-ERROR	Action
Other keys.[4]	NO	Beep terminal.
Other keys.[4]	YES	Return control to procedure.

[1] Valid cursor motion keys within a frame are CURSOR UP, CURSOR DOWN, CURSOR RIGHT, CURSOR LEFT, SPACEBAR, TAB, and BACKTAB.

[2] Invalid cursor motion keys are CURSOR UP, CURSOR DOWN, CURSOR RIGHT, and CURSOR LEFT that cause the cursor to move outside the frame.

[3] The `r-chs1.p` procedure below, shows what the CHOOSE statement does when the user enters a non-unique string followed by a character that, together with the rest of the string, does not match anything.

[4] Other keys are non-cursor-motion, non-alphanumeric keys (function keys, BACKSPACE) except for: HELP, STOP, RETURN, GO, END, ERROR, END-ERROR. Keys defined to do the actions of these keys still do so.

r-chs1.p

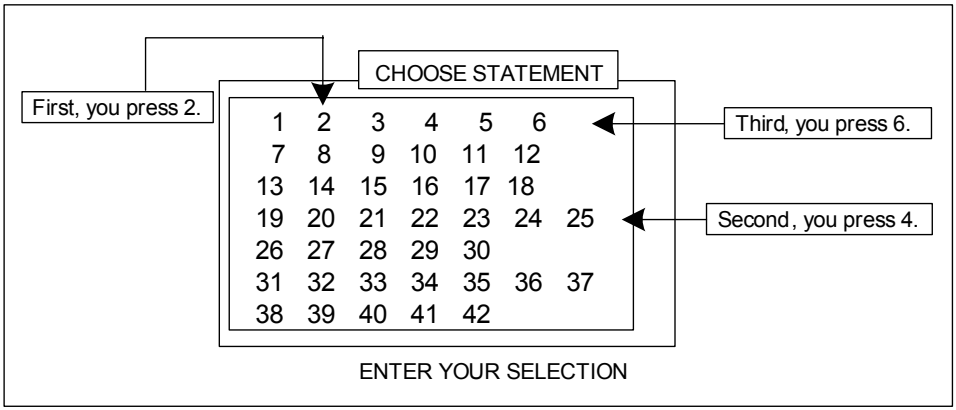
```
DEFINE VARIABLE abc AS CHARACTER FORMAT "x(3)" EXTENT 42.
DEFINE VARIABLE i AS INTEGER.

DO i = 1 TO 42:
  abc[i] = STRING(i,">9").
END.

DISPLAY abc NO-LABELS WITH ATTR-SPACE CENTERED ROW 4
  TITLE " CHOOSE STATEMENT " FRAME f-choose WIDTH 36.
DISPLAY "Enter your selection " WITH CENTERED NO-BOX
  FRAME f-instruct.
PAUSE 1 BEFORE-HIDE NO-MESSAGE.

REPEAT:
  HIDE MESSAGE.
  CHOOSE FIELD abc AUTO-RETURN WITH FRAME f-choose.
  MESSAGE "You selected -> " FRAME-VALUE.
END.
```

Once you run this procedure, your window looks like the following:



- When you press 2, CHOOSE moves the highlight bar to 2. When you press 4, CHOOSE moves the bar to 24. When you press 6, CHOOSE looks for the string 246. Because it cannot find the string, it matches the last key pressed (6) and places the highlight bar on 6.
- A choose field can temporarily become a handle type for internal purposes, but is not actually a widget since it does not have its own set of attributes and widgets. Therefore, you might see myhandle:TYPE = choose field in the widget tree, but you cannot manipulate the choose field.

See also COLOR phrase, Frame phrase, SCROLL statement, STATUS statement

CHR function

Converts an integer value to its corresponding character value.

Syntax

```
CHR ( expression
      [ , target-codepage [ , source-codepage ] ]
    )
```

expression

An expression that yields an integer value that you want to convert to a character value.

If the value of *expression* is in the range of 1 to 255, CHR returns a single character. This character might not be printable or might not display on certain terminals. For a value greater than 255 and less than 65535, the CHR function checks for a corresponding lead-byte value. If the integer value corresponds to a valid lead-byte, the CHR returns a double-byte character.

The CHR function returns a null string if the *expression* yields a value outside of the range 1 to 65534 or the *expression* yields a value in the range 256 to 65534 and the value does not correspond to a valid lead-byte.

target-codepage

A character-string expression that evaluates to the name of a code page. The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management). If you supply a non-valid name, the CHR function returns a null string. Before returning a character value, the CHR function converts *expression* from *source-codepage* to *target-codepage*. The returned character value is relative to *target-codepage*. If you do not specify *target-codepage*, no code page conversions occur.

source-codepage

A character-string expression that evaluates to the name of a code page. The name that you specify must be a valid code page name available in the DLC/convmap.cp file. If you supply a non-valid name, the CHR function returns a null string. The *source-codepage* specifies the name of the code page to which *expression* is relative. The default value of *source-codepage* is the value of SESSION:CHARSET.

Example This procedure initializes the 26 elements of the letter array to the letters A through Z:

r-chr.p

```
DEFINE VARIABLE letter AS CHARACTER FORMAT "X(1)" EXTENT 26.
DEFINE VARIABLE I AS INTEGER.

DO I = 1 TO 26:
  letter[i] = CHR((ASC("A")) - 1 + i).
END.
DISPLAY SKIP(1) letter WITH 2 COLUMNS NO-LABELS
  TITLE "T H E  A L P H A B E T".
```

Notes

- The CHR function returns the corresponding character in the specified code page. By default, the value of SESSION:CHARSET is iso8859-1. You can set a different internal code page by specifying the Internal Code Page (-cpinternal) parameter. For more information, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- The CHR function is double-byte enabled. For a value greater than 255 and less than 65535, it checks for a lead-byte value. If the lead-byte value is valid, Progress creates and returns a double-byte character.

See also ASC function, CODEPAGE-CONVERT function, SESSION system handle, STRING function

CLASS statement

Defines a user-defined class, as well as the data members and methods that comprise the class.

Note: This statement is applicable only when used in a class definition (`.cls`) file.

Syntax

```
CLASS type-name [ INHERITS super-type-name ]
  [ IMPLEMENTS interface-type-name [, interface-type-name ] ... ]
  [ USE-WIDGET-POOL ]
  [ FINAL ]:

  class-body
```

type-name

A character string that specifies the type name of the class. Specify a class type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

INHERITS *super-type-name*

An optional character string that specifies the type name of a super class whose state and behavioral characteristics the new class inherits. The new class becomes a subclass of this super class. Specify a super class type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

The class definition (`.cls`) file for the super class must be found at compile time.

IMPLEMENTS *interface-type-name* [, *interface-type-name*] ...

An optional character string that specifies the type name of one or more interfaces the new class implements. The new class must implement all methods defined in the specified interfaces. Specify an interface type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

The class definition (`.cls`) file for the interface must be found at compile time.

For more information about defining an interface, see the INTERFACE statement reference entry in this book.

`USE-WIDGET-POOL`

Directs Progress to create an unnamed widget pool scoped to this class. When specified, all dynamic object instances are created in the unnamed widget-pool by default. Otherwise, object instances are created in the session unnamed widget-pool. Progress deletes the unnamed widget-pool when the object instance is deleted.

You can also define a named or unnamed widget-pool in a class definition file. If you define a named widget-pool, object instances are created in that widget-pool only if you explicitly reference the widget-pool name.

When you define an unnamed widget-pool in a method, it is scoped to that method. You can delete the widget-pool within the method by using the DELETE WIDGET-POOL statement, or let Progress delete the widget-pool when the method ends.

For more information about defining and using widget-pools within a class, see *OpenEdge Getting Started: Object-oriented Programming*.

`FINAL`

Indicates the class cannot be inherited by another class. That is, it cannot be used in the INHERITS phrase in another class definition. Define a class as FINAL when you do not want any of its state or behavior overridden.

`class-body`

The body of a class definition is composed of the following types of elements:

- Data members, which define the instance data or state of the class.
- Constructor method, which is invoked when the object is instantiated.
- Methods, which define the behavior of the class.
- Destructor method, which is invoked when the object is deleted.

You can define data members, class-specific methods, and the constructor and destructor methods in any order.

You can also specify ON statements within the body of a class definition.

Define elements in the class body using the following syntax:

```
[ data-member-definitions ]
[ constructor-definition ]
[ method-definitions ]
[ destructor-definition ]
END [ CLASS ].
```

data-member-definitions

Defines one or more data members in the class. Table 12 lists the types of data members you can define and their associated Progress 4GL DEFINE statement.

Table 12: Class data member types *(1 of 2)*

Data member type	Progress 4GL DEFINE statement
BROWSE widget	DEFINE BROWSE statement
BUFFER object	DEFINE BUFFER statement
BUTTON widget	DEFINE BUTTON statement
CLASS object	DEFINE VARIABLE statement with the AS CLASS option (defines a variable as an object reference for a class or an interface)
DATASET object	DEFINE DATASET statement
DATA-SOURCE object	DEFINE DATA-SOURCE statement
FRAME widget	DEFINE FRAME statement
HANDLE	DEFINE VARIABLE statement
IMAGE widget	DEFINE IMAGE statement
MENU widget	DEFINE MENU statement
QUERY object	DEFINE QUERY statement
RECTANGLE widget	DEFINE RECTANGLE statement

Table 12: Class data member types *(2 of 2)*

Data member type	Progress 4GL DEFINE statement
STREAM	DEFINE STREAM statement
SUB-MENU widget	DEFINE SUB-MENU statement
TEMP-TABLE object	DEFINE TEMP-TABLE statement
VARIABLE	DEFINE VARIABLE statement (defines a variable as a data member for a class)
WIDGET-HANDLE	DEFINE VARIABLE statement
WORK-TABLE widget	DEFINE WORK-TABLE statement

For more information about these data member types, and any class-related restrictions, see the associated Progress 4GL DEFINE statement reference entry in this book.

constructor-definition

Defines a constructor method for the class using the CONSTRUCTOR statement. Progress invokes this constructor method to initialize data or state for a new class object instance when the object is instantiated using the NEW statement. You can define only one constructor method for a class. You cannot invoke a constructor method directly. If not defined, Progress provides a default constructor method that takes no parameters. For more information, see the CONSTRUCTOR statement reference entry in this book.

method-definitions

Defines one or more methods in the class. For more information, see the METHOD statement reference entry in this book.

destructor-definition

Defines a destructor method for the class using the DESTRUCTOR statement. Progress invokes this destructor method when the object is deleted using the DELETE OBJECT statement. You cannot invoke a destructor method directly. If not defined, Progress provides a default destructor method. For more information, see the DESTRUCTOR statement reference entry in this book.

END [CLASS]

Specifies the end of the class body definition. You must end the class body definition with the END statement.

Example

The following example shows the definition of a class that inherits a super class and defines several data members:

```
CLASS acme.myObjs.CustObj INHERITS acme.myObjs.Common.CommonObj:

  DEFINE PUBLIC VARIABLE m_NumCusts AS INTEGER NO-UNDO.
  DEFINE PROTECTED TEMP-TABLE ttCust NO-UNDO
     FIELD CustNum LIKE Customer.CustNum
     FIELD Name LIKE Customer.Name.
  DEFINE PRIVATE BUFFER cust FOR Customer.

END CLASS.
```

Notes

- You can terminate a CLASS statement with either a period (.) or a colon (:).
- A complete class definition must begin with the CLASS statement and end with the END statement.
- The access mode for a class definition is always PUBLIC.
- A class definition (`.cls`) file can contain only one class definition.
- In effect, a user-defined class represents a unique data type. In Progress, you can use a class just as you would any Progress built-in data type. You can define variables, parameters and return types as a class. You can also assign an object reference for a class object instance to a temporary table field defined as a Progress.Lang.Object; but you cannot assign an object reference to a field in a database table.

- You can reference include files from within a class definition file. For more information about include files, see the { } Include file reference entry in this book.
- All built-in preprocessor directives are supported in class definition files.
- All built-in preprocessor names are supported in class definition files. For a list of preprocessor name, see the { } Preprocessor name reference entry in this book.
- You cannot pass compile-time arguments to class definition files. However, you can pass compile-time arguments to include files referenced in a class definition file.
- The compiled version of a class definition file is an r-code (`.r`) file. For more information, see the COMPILE statement reference entry in this book.
- You cannot run r-code compiled for a class definition file with the RUN statement.
- You cannot define a data member, of any data type, in a subclass using the same name as a PUBLIC or PROTECTED data member in one of its super classes.
- PUBLIC and PROTECTED variables defined as data members within a class definition (`.cls`) file maintain their characteristic throughout the inherited class hierarchy. Do not repeat variable definitions in any inherited class definition files.
- You create an object instance of a class, and assign an object reference to the instance, using the NEW statement. You access a class object instance, as well as its data members and methods, using its associated object reference. For more information about the NEW statement, see the NEW statement reference entry in this book.
- Progress provides a system reference for the currently running object instance of this class, called THIS-OBJECT. For more information, see the THIS-OBJECT system reference entry in this book.
- If this class is a subclass of some super class, you can use the SUPER system reference to access the PUBLIC and PROTECTED methods of all super classes within the inherited class hierarchy. For more information, see the SUPER system reference entry in this book.
- You can store class definition r-code files in Progress procedure libraries. If Progress encounters a procedure library on `PROPATH`, it will search the library for the specified r-code. However, you cannot execute r-code files stored in a procedure library that is not on `PROPATH` using the *`procedure-library-path<<member-name>>`* syntax.

- You cannot define a NEW SHARED or NEW GLOBAL SHARED variable in a class definition (`.cls`) file.
- You cannot use ActiveX controls within a class definition (`.cls`) file.
- You can use COM automation objects within a class definition (`.cls`) file. However, any event handlers for the object must be defined in a procedure file.
- You cannot define an array of classes.
- For more information about class definition (`.cls`) files, see *OpenEdge Getting Started: Object-oriented Programming*.

See also CONSTRUCTOR statement, DEFINE PARAMETER statement, DEFINE VARIABLE statement, DESTRUCTOR statement, INTERFACE statement, METHOD statement, NEW statement

CLEAR statement

Clears the data and colors (and side labels for a down frame) for all fill-in fields in a frame.

Note: Does not apply to SpeedScript programming.

Syntax

```
CLEAR [ FRAME frame ] [ ALL ] [ NO-PAUSE ]
```

FRAME *frame*

Represents the name of the frame containing the fill-in fields you want to clear. If you do not name a frame, CLEAR clears the default frame for the block containing the CLEAR statement.

ALL

Clears all occurrences and resets the current display position to the top of the frame for a down frame (a frame used to display several occurrences of the fields in the frame).

NO-PAUSE

Does not pause before clearing the frame.

Example

The `r-clear.p` procedure displays the Progress data types and their corresponding default formats. The procedure prompts you to enter values so you can see how Progress formats those values. If you answer YES, Progress clears the values currently displayed so that you can enter new values.

r-clear.p

```
DEFINE VARIABLE a AS CHARACTER INITIAL "xxxxxxxx".
DEFINE VARIABLE b AS DATE INITIAL TODAY.
DEFINE VARIABLE c AS DECIMAL INITIAL "-12,345.67".
DEFINE VARIABLE d AS INTEGER INITIAL "-1,234,567".
DEFINE VARIABLE e AS LOGICAL INITIAL yes.

DISPLAY "This illustrates the default formats for the
  different data types" SKIP (2) WITH CENTERED
  ROW 4 NO-BOX FRAME head.
FORM "CHARACTER default format is ""x(8)""    " a SKIP
  "DATE default format is 99/99/99         " b SKIP
  "DECIMAL default format is ->>,>>9.99    " c SKIP
  "INTEGER default format is ->,>>>,>>9    " d SKIP
  "LOGICAL default format is yes/no        " e TO 55 SKIP
  WITH ROW 8 NO-BOX NO-LABELS CENTERED FRAME ex.
REPEAT:
  DISPLAY a b c d WITH FRAME ex.
  MESSAGE "Do you want to put in some values?"
  UPDATE e.
  IF e THEN DO:
    CLEAR FRAME ex NO-PAUSE.
    SET a b c d WITH FRAME ex.
  END.
  ELSE LEAVE.
END.
```

Notes

- The CLEAR statement only clears fill-in fields. GUI widgets such as editors or radio-sets are not affected.
- Progress automatically clears a single (1 down) frame whenever its block is iterated. Progress automatically clears a multi-frame (down frame) whenever it is full and its block where it is iterated.

CLOSE QUERY statement

Closes a query that was opened by a previous OPEN QUERY statement.

Syntax

```
CLOSE QUERY query
```

query

The name of an open query.

Example

The `r-clsqry.p` procedure defines a query, q-cust, which it shares with `r-query.p`. Each time you choose the Ascending, Descending, or Cust-Num button, the procedure opens a new query for q-cust. To do this, the procedure must first close an open query for each q-cust. Therefore, the CLOSE QUERY statement is used in the CHOOSE trigger for each of these buttons.

r-clsqry.p

```
DEFINE NEW SHARED BUFFER x-cust FOR customer.
DEFINE NEW SHARED QUERY  q-cust FOR x-cust.

DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE QUIT.
  END.

DEFINE BUTTON b_ascend  LABEL "Ascending".
DEFINE BUTTON b_descend LABEL "Descending".
DEFINE BUTTON b_num     LABEL "Cust-Num".

FORM b_ascend b_descend b_num b_quit
  WITH FRAME butt-frame ROW 1.

ON CHOOSE OF b_ascend
DO:
  CLOSE QUERY q-cust.
  OPEN QUERY q-cust FOR EACH x-cust NO-LOCK
    BY x-cust.name.
  DISABLE ALL WITH FRAME butt-frame.
  RUN r-query.p.
END.

ON CHOOSE OF b_descend
DO:
  CLOSE QUERY q-cust.
  OPEN QUERY q-cust FOR EACH x-cust NO-LOCK
    BY x-cust.name DESCENDING.
  DISABLE ALL WITH FRAME butt-frame.
  RUN r-query.p.
END.

ON CHOOSE OF b_num
DO:
  CLOSE QUERY q-cust.
  OPEN QUERY q-cust FOR EACH x-cust NO-LOCK
    BY x-cust.cust-num.
  DISABLE ALL WITH FRAME butt-frame.
  RUN r-query.p.
END.

DO WHILE TRUE:
  ENABLE ALL WITH FRAME butt-frame.
  WAIT-FOR CHOOSE OF b_ascend, b_descend, b_num, b_quit.
END.
```

r-query.p

```
DEFINE SHARED BUFFER x-cust FOR customer.
DEFINE SHARED QUERY  q-cust FOR x-cust.

GET FIRST q-cust.

DO WHILE AVAILABLE(x-cust):
  DISPLAY x-cust.name
          x-cust.cust-num
    WITH FRAME cust-info CENTERED DOWN ROW 3 USE-TEXT.
  DOWN 1 WITH FRAME cust-info.
  GET NEXT q-cust.
END.
```

Notes

- If a query is closed, you cannot retrieve any more records for the query.
- Closing a query frees most resources used by the query.
- After you close a query, you can reopen it with the OPEN QUERY statement. However, you cannot reuse the query's buffers for a different table. For example, a buffer, buff1, is created for the customer table in a DEFINE QUERY or OPEN QUERY for the query, qry1. The query is run and closed. You cannot now DEFINE or OPEN qry1 with buff1 for the item table. You can reuse buffers with CREATE QUERY, but you must re-run QUERY-PREPARE.
- If you do not explicitly close a query, it is closed when another OPEN QUERY statement is executed for the same query name.

See also

CURRENT-RESULT-ROW function, DEFINE QUERY statement, GET statement, NUM-RESULTS function, OPEN QUERY statement, REPOSITION statement

CLOSE STORED-PROCEDURE statement

For a non-Progress stored procedure, indicates that the procedure has completed execution and retrieves any return status. For a send-sql-statement stored procedure, closes the SQL cursor used by the procedure.

Syntax

```
CLOSE STORED-PROCEDURE procedure
  [ integer-field = PROC-STATUS ]
  [ WHERE PROC-HANDLE = integer-field ]
```

procedure

The name of the stored procedure that you want to close or the built-in procedure name, send-sql-statement.

integer-field = PROC-STATUS

Assigns the return value from a stored procedure to the specified integer field or variable (*integer-field*).

WHERE PROC-HANDLE = *integer-field*

An integer field or variable whose value uniquely identifies the stored procedure that produces the results returned from the data source or the SQL cursor of a send-sql-statement stored procedure.

Example

The PROC-STATUS clause of the CLOSE STORED-PROCEDURE statement allows the DataServer for ORACLE to retrieve the text of an ORACLE error message that was passed to raise_application_error. Use the ERROR-STATUS:GET-MESSAGE handle to retrieve the message, as shown in the following example:

```
DEFINE VARIABLE st AS INTEGER INITIAL 0.
DEFINE VARIABLE h  AS INTEGER.

RUN STORED-PROC p1 h = PROC-HANDLE NO-ERROR.

CLOSE STORED-PROC p1 st = PROC-STATUS WHERE PROC-HANDLE = h.

DISPLAY st.

IF ERROR-STATUS:ERROR
     THEN
   MESSAGE ERROR-STATUS:GET-MESSAGE(1) ERROR-STATUS:GET-NUMBER(1)
          VIEW-AS ALERT-BOX.
```

Notes

- If you specified a PROC-HANDLE when you ran a stored procedure, you must specify the PROC-HANDLE when you close the stored procedure.
- If you do not specify a PROC-HANDLE, the CLOSE STORED-PROCEDURE statement will close the procedure if there is only one stored procedure running. If there is more than one stored procedure running, an error will be returned.
- You cannot close a send-sql-statement procedure until you have retrieved all row results.
- You can close all stored procedures at once with the following statement:

```
RUN STORED-PROC closeallprocs.
```

- For more information on using this statement, see *OpenEdge Data Management: DataServer for ODBC* and *OpenEdge Data Management: DataServer for ORACLE*.

See also

PROC-HANDLE function, PROC-STATUS function, RUN STORED-PROCEDURE statement

CODEPAGE-CONVERT function

Converts a string value from one code page to another.

Syntax

```
CODEPAGE-CONVERT
  ( source-string
    [ , target-codepage [ , source-codepage ] ]
  )
```

source-string

A CHARACTER or LONGCHAR expression to be converted.

target-codepage

A character-string expression that evaluates to the name of a code page. The returned character value is relative to *target-codepage*. The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management). If you supply a non-valid name, the CODEPAGE-CONVERT function returns the Unknown value (?). Before returning a character value, the CODEPAGE-CONVERT function converts *source-string* from *source-codepage* to *target-codepage*. If you do not specify *target-codepage*, no code page conversions occur.

source-codepage

A character-string expression that evaluates to the name of a code page. The *source-codepage* specifies the name of the code page to which *source-string* is relative. The name that you specify must be a valid code page name available in the DLC/convmap.cp file. If you supply a non-valid name, the CODEPAGE-CONVERT function returns the Unknown value (?). The default value of *source-codepage* is the value of CHARSET attribute of the SESSION handle.

If *source-string* is a LONGCHAR variable, the *source-codepage* argument is not valid. In this case, the code page of the LONGCHAR variable is used as the source code page.

Example

This example assumes that the native code page of `r-codpag.p` is ibm850. It is written so that its embedded text strings are always converted to the internal code page of the OpenEdge session (SESSION:CHARSET).

r-codpag.p

```
DEFINE VARIABLE cp850string AS CHARACTER INITIAL "text with umlaut (ä)".
DEFINE VARIABLE charsetstring AS CHARACTER.

charsetstring = CODEPAGE-CONVERT(cp850string, SESSION:CHARSET, "ibm850").

FOR EACH item:
    IF LOOKUP(charsetstring, item.cat-description) > 0 THEN DO:
        DISPLAY item.item-name.
    END.
END.
```

Notes

- The CODEPAGE-CONVERT function returns the corresponding character string in the specified code page. By default, the value of SESSION:CHARSET is iso8859-1. You can set a different internal code page by specifying the Internal Code Page (-cpinternal) parameter. For more information, see *OpenEdge Development: Internationalizing Applications* and *OpenEdge Deployment: Startup Command and Parameter Reference*.
- This function is especially useful if you plan to run a procedure in an OpenEdge session in which the SESSION:CHARSET code page is different from the native code page of the procedure.
- When you write procedures with the Progress 4GL, you must use 7-bit (that is, ASCII) characters for field names and variable names. But you can use 8-bit and multi-byte characters, including Unicode, for data values such as character strings and constants. Thus, a procedure written and compiled on a system using one code page can be run on a system using another code page as long as you convert all embedded character strings to the internal code page. Using CODEPAGE-CONVERT as shown in the example allows your procedures to be virtually code page independent.

See also

ASC function, CHR function, STRING function

COLOR phrase

Specifies a video attribute or color. In Progress Version 7 and later, the COLOR phrase is superseded by the FGCOLOR and BGCOLOR options in graphical user interfaces and by the PFCOLOR and DCOLOR options in character interfaces. The COLOR phrase is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
{    NORMAL
   | INPUT
   | MESSAGES
   | protermcap-attribute
   | dos-hex-attribute
   | { [ BLINK- ] [ BRIGHT- ]
       [ fgnd-color ] [ bgnd-color ]
     }
   | { [ BLINK- ] [ RVV- ] [ UNDERLINE- ] [ BRIGHT- ]
       [ fgnd-color ]
     }
   | VALUE ( expression )
}
```

NORMAL, INPUT, MESSAGES

The three standard colors Progress uses for screen displays. Progress uses NORMAL to display fields, INPUT to display input fields, and MESSAGES to display items in the message area.

Following are the NORMAL defaults:

- **Windows** — On a color monitor, the default colors are a blue background and a white foreground. On a monochrome monitor, the default colors are a standard background and foreground, depending on the monitor.
- **UNIX** — The default colors are the normal display mode of your terminal.

Following are the INPUT defaults:

- **Windows** — On a color monitor, the default colors are a light gray background and a blue foreground. On a monochrome monitor, the default underlines fields that require input.
- **UNIX** — The default colors depend on the type of terminal and how INPUT is defined in the protermcap file, but it is usually underlining.

Following are the MESSAGES defaults:

- **Windows** — On a color monitor, the defaults are the same as for INPUT. On a monochrome monitor, the default is reverse video.
- **UNIX** — The defaults depend on the type of terminal and how MESSAGES is defined in the protermcap file, but it is usually reverse video. (The protermcap file supplied with Progress supplies default attributes for NORMAL, INPUT, and MESSAGES for all defined terminals.)

protermcap-attribute

You use the *protermcap-attribute* option only if you are using UNIX. This is the name assigned to the attribute in the *protermcap* file (for example, RED, BLINK, etc.). See *OpenEdge Deployment: Managing 4GL Applications* for a description of the *protermcap* file.

dos-hex-attribute

A hex string with a value of 00 through FF.

[BLINK- **]** **[** BRIGHT- **]** **[** *fgnd-color* **]** **[** *bgnd-color* **]**

Names specific colors you want to use for the screen foreground and background. You use this option only if you are using Windows, and usually only if you use a color monitor. Table 13 lists the colors you can use for *fgnd-color* and *bgnd-color*.

Table 13: Windows colors

Color	Abbreviation
Black	Bla, Blk
Blue	Blu
Green	Gre, Grn
Cyan	C
Red	Red
Magenta	Ma
Brown	Bro, Brn
Gray	Gra, Gry
Dark-Gray	D-Gra
Light-Blue	Lt-Blu
Light-Green	Lt-Gre
Light-Cyan	Lt-C
Light-Red	Lt-Red
Light-Magenta	Lt-Ma
Light-Brown	Lt-Bro
Yellow	Y
White	W

If *fgnd-color* is omitted, then the system uses the foreground corresponding to NORMAL. If *bgnd-color* is omitted, then the system uses the background corresponding to NORMAL. If NORMAL, INPUT, or MESSAGES is specified for *fgnd-color* or *bgnd-color*, then the system uses the foreground or background color of the specified standard color.

[BLINK- **]** **[** RVV- **]** **[** UNDERLINE- **]** **[** BRIGHT- **]** **[** *fgnd-color* **]**

Names specific attributes you want to use for the screen display. Use this option only if you are using Windows, and usually only if you use a monochrome monitor. Normally, you would never specify *fgnd-color*.

VALUE (*expression*)

An expression with a value that results in one of the options in the COLOR phrase.

Example

The following procedure displays a random number of asterisks, in a random color, column, and row in 10 different occurrences. The COLOR statement displays the asterisks in one of the three colors stored in the elements of the hilite array. The COLOR phrase in this example is VALUE (hilite[RANDOM(1,3)]. The DISPLAY statement uses the color determined in the COLOR statement to display a random number of asterisks.

r-colphr.p

```
DEFINE VARIABLE hilite AS CHARACTER EXTENT 3.
DEFINE VARIABLE loop AS INTEGER.
hilite[1] = "NORMAL".
hilite[2] = "INPUT".  /* attribute to highlight */
hilite[3] = "MESSAGES".

REPEAT WHILE loop <= 10:
  FORM bar AS CHARACTER WITH ROW(RANDOM(3,17))
    COLUMN(RANDOM(5,50)) NO-BOX NO-LABELS
    FRAME bursts.
  COLOR DISPLAY VALUE(hilite[RANDOM(1,3)]) bar
    WITH FRAME bursts.
  DISPLAY FILL("*",RANDOM(1,8)) @ bar WITH FRAME bursts.
  PAUSE 1 NO-MESSAGE.
  HIDE FRAME bursts NO-PAUSE.
  loop = loop + 1.
END.
```

Note

- For an application to use this COLOR phrase, it must use the default color table in the installed environment.
- The system ignores the color phrase entry for overlay frames on spacetaking terminals.
- For more information on the `protermcap` file, see *OpenEdge Deployment: Managing 4GL Applications*.

See also COLOR statement

COLOR statement

Indicates the video attribute or color for normal display or for data entry.

Note: Does not apply to SpeedScript programming.

Syntax

```
COLOR [ DISPLAY ] color-phrase [ PROMPT color-phrase ]
      { field ... } { [ frame-phrase ] }
```

```
COLOR PROMPT color-phrase
      { field ... } { [ frame-phrase ] }
```

DISPLAY

Indicates that you want to use a specific color when the system displays a field.

PROMPT

Indicates that you want to use a specific color when the system prompts a user for input by an INSERT, PROMPT-FOR, SET, or UPDATE statement.

color-phrase

Specifies a video attribute or color. Following is the syntax for *color-phrase*:

```
{     NORMAL
    | INPUT
    | MESSAGES
    | protermcap-attribute
    | dos-hex-attribute
    | { [ BLINK- ] [ BRIGHT- ]
        [ fgnd-color ] [ bgnd-color ]
      }
    | { [ BLINK- ] [ RVV- ] [ UNDERLINE- ] [ BRIGHT- ]
        [ fgnd-color ]
      }
    | VALUE ( expression )
}
```

For more information on *color-phrase*, see the COLOR phrase reference entry. Progress ignores the *color-phrase* entry for overlay frames on spacetaking terminals.

field

The name of the field or fields for which you want to override the default colors.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information see the Frame phrase reference entry.

Example

This procedure highlights the item number and on-hand fields for items with an on-hand value less than 50. The variable hilite holds the video attribute (color) for highlighting. In this case, the system uses whatever attribute is used for the message area (such as reverse video, bright, or a color).

r-color.p

```
DEFINE VARIABLE hilite AS CHARACTER.hilite = "messages".

  /* Use standard messages attribute to
    highlight on-hand less than 50 */
FOR EACH item:
  DISPLAY item-num item-name on-hand WITH ATTR-SPACE.
  IF on-hand < 50 THEN
    COLOR DISPLAY VALUE(hilite) item-num on-hand.
END.
```

Notes

- When the output destination is not the terminal, Progress disregards the COLOR statement.
- The COLOR statement does not automatically display a frame whose field's color attribute is changing.
- Use one of these statements to reset a field to the Progress default colors:

```
COLOR DISPLAY NORMAL PROMPT INPUT field
```

Or

```
COLOR DISPLAY NORMAL PROMPT INPUT field WITH FRAME frame
```

- If you run precompiled procedures on a spacetaking terminal, you must specify the frame field where a color or other video attribute is applied as, or is by default, ATTR-SPACE.

- If you write a procedure (for a non-spacetaking terminal) that uses color and you run it on a spacetaking terminal, Progress does not display the colors. To display the colors, you must use the ATTR-SPACE option.
- Certain terminals, such as the WYSE 75, are non-spacetaking for some attributes and spacetaking for others.
- On UNIX, if you specify a color or video attribute that is not defined for the terminal, Progress uses normal display instead.

See also COLOR phrase, DISPLAY statement, Frame phrase

COMBO-BOX phrase

Describes a combo-box widget. A combo-box represents a field or variable, and consists of a field value and an associated drop-down list of possible values.

Note: Does not apply to SpeedScript programming.

Syntax

```
COMBO-BOX
  [ LIST-ITEMS item-list | LIST-ITEM-PAIRS item-pair-list ]
  [ INNER-LINES lines ] [ size-phrase ] [ SORT ]
  [ TOOLTIP tooltip ]
  [ SIMPLE | DROP-DOWN | DROP-DOWN-LIST ]
  [ MAX-CHARS characters ]
  [ AUTO-COMPLETION [ UNIQUE-MATCH ] ]
```

LIST-ITEMS *item-list*

Specifies the items to appear in the drop-down list. *item-list* represents a comma-separated list of valid values for the field or variable.

LIST-ITEM-PAIRS *item-pair-list*

Specifies a list of label-value pairs. Each pair represents the label and value of a field or variable. When the drop-down list appears, it displays each pair's label. Then, if the user selects a label, Progress assigns the corresponding value to the field or variable. The syntax for *item-pair-list* is as follows:

```
label , value [ , label , value ] ...
```

label

A character string representing the label of the field or variable.

value

A value that Progress assigns to the field or variable if the user selects the corresponding label.

INNER-LINES *lines*

Specifies the number of lines visible in the drop-down list for a DROP-DOWN or DROP-DOWN-LIST combo-box widget. The value for *lines* must be 3 or greater. If the number of lines you specify is less than the number of items in the drop-down list, the list is scrollable.

The INNER-LINES option in a SIMPLE combo-box definition is ignored.

size-phrase

Specifies the outside dimensions (width and height) of the combo-box widget and its drop-down list using the SIZE phrase. You must specify a SIZE phrase in the definition of a SIMPLE or DROP-DOWN combo-box widget. The syntax for the SIZE phrase is as follows:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information, see the SIZE phrase reference entry.

Note: The *height* value is ignored for DROP-DOWN and DROP-DOWN-LIST combo-box widgets. The height is always set to the height of a fill-in for the current font.

SORT

Specifies that list items be sorted prior to display.

TOOLTIP *tooltip*

Allows you to define a help text message for a text field or text variable. Progress automatically displays this text when the user pauses the mouse button over a text field or text variable for which a tooltip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to “” or the Unknown value (?), then the tooltip is removed. No tooltip is the default. The TOOLTIP option is supported in Windows only.

SIMPLE

Specifies a combo-box widget with a read/write edit control and a list that is always visible. This option is supported in graphical interfaces only, and only in Windows. If you specify a SIMPLE combo-box widget in a character interface, Progress treats it as a DROP-DOWN-LIST combo-box widget.

DROP-DOWN

Specifies a combo-box widget with a read/write edit control and a drop-down list that appears when you click the drop-down button. This option is supported in graphical interfaces only, and only in Windows. If you specify a DROP-DOWN combo-box widget in a character interface, Progress treats it as a DROP-DOWN-LIST combo-box widget.

DROP-DOWN-LIST

Specifies a combo-box widget with a read-only edit control and a drop-down list that appears when you click the drop-down button. This is the default.

MAX-CHARS *characters*

The maximum number of characters the edit control can hold. The *characters* parameter must be a positive integer constant. If *characters* is zero or the Unknown value (?), MAX-CHARS is set to 255 characters by default.

Use MAX-CHARS with only SIMPLE and DROP-DOWN combo-boxes. It is ignored for DROP-DOWN-LIST combo-boxes. This option is supported in graphical interfaces only, and only in Windows.

AUTO-COMPLETION

Specifies that the edit control automatically complete keyboard input to the combo-box, based on a potential match, by searching through the items in the drop-down list. This option is supported in graphical interfaces only, and only in Windows.

UNIQUE-MATCH

Specifies that the edit control complete keyboard input to the combo-box based on a unique match. This option is supported in graphical interfaces only, and only in Windows.

Examples

The first example, `r-combo.p`, views a date field as a combo-box. When you run this procedure, you can choose a date value from the drop-down list. When you choose a new value, the VALUE-CHANGED trigger updates the value of out-string to an event associated with the new date value.

The example initializes the drop-down list by building a comma-separated list of values and then assigning the string to the LIST-ITEMS attribute of the combo-box.

r-combo.p

```
DEFINE VARIABLE hist-date AS DATE FORMAT "99/99/9999"
        VIEW-AS COMBO-BOX
        LIST-ITEMS 07/04/1776, 07/11/1969, 09/10/1993.
DEFINE VARIABLE hist-event AS CHARACTER INITIAL
  "Declaration of Independence,Man walks on moon,Progress Version 7
  ships".
DEFINE VARIABLE out-string AS CHARACTER FORMAT "x(36)".

DEFINE FRAME main-frame
   hist-date out-string
   WITH NO-LABELS TITLE "Historic Events".

ON VALUE-CHANGED OF hist-date
   DO:
      out-string = ENTRY(SELF:LOOKUP(SELF:SCREEN-VALUE), hist-event).
      DISPLAY out-string WITH FRAME main-frame.
   END.

ENABLE hist-date WITH FRAME main-frame.

APPLY "VALUE-CHANGED" TO hist-date IN FRAME main-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

The following example, `r-combo.p`, builds a combo-box based on field values from a database. It defines triggers that allow you to change the value of the combo-box without displaying the drop-down list. They allow you to scroll through the values using the **CURSOR-DOWN** and **CURSOR-UP** keys or to jump to a specific value by typing its first letter.

r-combo2.p

```
DEFINE VARIABLE i AS INTEGER.
DEFINE VARIABLE rep AS CHARACTER LABEL "Rep" VIEW-AS COMBO-BOX.
DEFINE VARIABLE temp-string AS CHARACTER.
FORM
   rep
   WITH FRAME main-frame SIDE-LABELS.
ON ANY-PRINTABLE OF rep
   DO:
      /* Find the first entry in the drop-down list
         that begins with the character typed. Set
         the SCREEN-VALUE of the combo box to that value. */
      seek-item:
      DO i = 1 TO SELF:NUM-ITEMS:
         IF SELF:ENTRY(i) BEGINS LAST-EVENT:FUNCTION
         THEN DO:
             SELF:SCREEN-VALUE = SELF:ENTRY(i).
             LEAVE seek-item.
         END.
      END.
      IF i > SELF:NUM-ITEMS
      THEN BELL.
   END.
ON CURSOR-DOWN OF rep
   DO:
      /* Change the SCREEN-VALUE of the combo box
         to the next value from the drop-down list. */
      i = SELF:LOOKUP(SELF:SCREEN-VALUE).
      IF i < SELF:NUM-ITEMS
      THEN SELF:SCREEN-VALUE = SELF:ENTRY(i + 1).
   END.
ON CURSOR-UP OF rep
   DO:
      /* Change the SCREEN-VALUE of the combo box
         to the prev value from the drop-down list. */
      i = SELF:LOOKUP(SELF:SCREEN-VALUE).
      IF i > 1
      THEN SELF:SCREEN-VALUE = SELF:ENTRY(i - 1).
   END.temp-string = "".
FOR EACH Salesrep NO-LOCK:
   IF temp-string = ""
   THEN temp-string = Salesrep.sales-rep.
   ELSE temp-string = temp-string + "," + Salesrep.sales-rep.
END.
ASSIGN rep:LIST-ITEMS IN FRAME main-frame = temp-string.
ENABLE rep WITH FRAME main-frame.
```

Notes

- When the drop-down list appears, if it contains the value associated with the field or variable, that value is initially highlighted. Otherwise, no value in the drop-down list is initially highlighted.
- The LIST-ITEMS option of the COMBO-BOX phrase requires a list of items (possibly quoted, depending on the combo-box's data type), such as ("a", "b", "c"), whereas the LIST-ITEMS attribute of a combo-box requires a quoted list of items, such as ("a, b, c"). Similarly, the LIST-ITEM-PAIRS option of the COMBO-BOX phrase requires a list of items (possibly quoted, depending on the combo-box's data type), such as ("a", 1, "b", 2, "c", 3), whereas the LIST-ITEM-PAIRS attribute of a combo-box requires a quoted list of items, such as ("a, 1, b, 2, c, 3").
- If you do not specify the LIST-ITEMS or LIST-ITEM-PAIRS option, the screen value of the variable or field becomes the null string (" "). To display or set values in the combo-box, you must first set the LIST-ITEMS or LIST-ITEM-PAIRS attribute to assign a drop-down list that specifies the available values.
- If you specify the SORT option for a COMBO-BOX, then any items you add with the ADD-FIRST, ADD-LAST, or INSERT methods are added in sorted order rather than the order you specify.
- Windows allows the user to transfer focus to the drop-down list by pressing **ALT** and one of the letters in the label. This is called a mnemonic.
- When you use the SIMPLE and DROP-DOWN options to define a character-field or character-variable combo-box widget, the FORMAT string for the field or variable is ignored.
- Items in a combo-box are case insensitive.

See also Format phrase, SIZE phrase, VIEW-AS phrase

COMPARE function

The COMPARE function compares two strings and lets you:

- Perform a raw compare, if desired.
- Use a particular collation.
- Turn case sensitivity on and off.

COMPARE returns a LOGICAL value representing the result of the logical expression, where the comparison rules are defined by the combination of the operator, the comparison strength, and the collation.

Syntax

```
COMPARE ( string1 , relational-operator , string2 ,
          strength [ , collation ] )
```

string1

A CHARACTER or LONGCHAR expression that evaluates to the first string to be compared.

relational-operator

A CHARACTER expression that evaluates to one of the relational operators, which are: LT (or <), LE (or <=), EQ (or =), GE (or >=), GT (or >), NE (or <>), “BEGINS”, and “MATCHES”.

string2

A CHARACTER or LONGCHAR expression that evaluates to the second string to be compared.

strength

A CHARACTER expression that evaluates to the Progress comparison strength or the International Components for Unicode (ICU) comparison strength to apply.

The Progress comparison strengths include:

- **RAW** — Progress compares the two strings using the numeric values in the current code page.
- **CASE-SENSITIVE** — Progress performs a case-sensitive comparison of the two strings using the numeric values in either the collation table specified in *collation*, or the collation table of the client. If you specify this strength with an ICU collation, Progress applies the ICU TERTIARY strength.
- **CASE-INSENSITIVE** — Progress performs a case-insensitive comparison of the two strings using the numeric values in either the collation table specified in *collation*, or the collation table of the client. If you specify this strength with an ICU collation, Progress applies the ICU SECONDARY strength.
- **CAPS** — Progress converts any lowercase letters in the two strings to uppercase letters, based on the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters, and then performs a raw comparison of the resulting strings. When neither string contains a wildcard character, this option behaves the same as the MATCHES operator.

The ICU comparison strengths include:

- **PRIMARY** — Progress compares the base characters in the two strings.
- **SECONDARY** — Progress compares the base characters and any diacritical marks in the two strings.
- **TERTIARY** — Progress performs a case-sensitive comparison of the base characters and diacritical marks in the two strings.

- **QUATERNARY** — Progress performs a case-sensitive comparison of the base characters and any diacritical marks in the two strings, and distinguishes words with and without punctuation. ICU uses this strength to distinguish between Hiragana and Katakana when applied with the ICU-JA (Japanese) collation. Otherwise, it is the same as TERTIARY.
- **IGNORE-SECONDARY** — Progress performs a case-sensitive, but diacritically-insensitive, comparison of the two strings.

Note: Use ICU comparison strengths only with ICU collations.

collation

A CHARACTER expression that evaluates to the name of a Progress collation table or ICU collation. By default, Progress uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).

If *`strength`* is not RAW or CAPS, the collation must be either a Progress collation table in the `convmap.cp` file or an ICU collation, and must be a valid collation table for the code page corresponding to the `-cpinternal` startup parameter.

Notes

- If either or both strings evaluate to the Unknown value (?), COMPARE returns the value indicated in Table 14.

Table 14: Relational operators and the Unknown value (?) *(1 of 2)*

Relational operator	Only one string evaluates to ?	Both strings evaluate to ?
LT (or <)	FALSE	FALSE
LE (or <=)	FALSE	TRUE
EQ (or =)	FALSE	TRUE
GE (or >=)	FALSE	TRUE

Table 14: Relational operators and the Unknown value (?) *(2 of 2)*

Relational operator	Only one string evaluates to ?	Both strings evaluate to ?
GT (or >)	FALSE	FALSE
NE (or <>)	TRUE	FALSE
BEGINS	FALSE	TRUE
MATCHES	FALSE	TRUE

- COMPARE returns the Unknown value (?) if one of the following occurs:
  - *relational-operator* does not evaluate to a valid value.
  - *strength* does not evaluate to a valid value.
  - *collation* does not evaluate to a collation table residing in the convmap.cp file.
  - *collation* evaluates to a collation table that is not defined for the code page corresponding to the -cpinternal startup parameter.
- LONGCHAR variable values are converted to -cpinternal for comparison and must convert without error, or Progress returns an error.
- With BEGINS, the language-sensitive rules are used only when *strength* is not RAW or CAPS.
- With MATCHES, CASE-SENSITIVE is treated as RAW, CASE-INSENSITIVE is treated as CAPS, and the collation is never used.

COMPILE statement

Compiles a procedure file or a class definition file. A compilation can last for a session, or you can save it permanently for use in later sessions (as an r-code file, which has a `.r` extension).

When you compile a class definition file, Progress compiles the class definition file identified in the COMPILE statement and all class files in its inherited class hierarchy, by default. You can direct Progress to compile only those class definition files in the class hierarchy that are not found in the cache, and cache any classes or interfaces it compiles during the session, by setting the MULTI-COMPILE attribute to TRUE.

Note: When you change the definition of a class, Progress Software Corporation recommends that you recompile all classes that inherit the modified class. This recommendation does not apply to method logic changes within a class.

After you compile a procedure file, you use the RUN statement to create an instance of the procedure, and you use a handle to access the procedure and its context. After you compile a class definition file, you use the NEW statement to create an object instance of the class, and you use an object reference to access the class object instance, as well as its data members and methods.

For more information about compiling procedure files, see *OpenEdge Development: Progress 4GL Handbook*. For more information about compiling class definition files, see *OpenEdge Getting Started: Object-oriented Programming*.

Syntax

```
COMPILE { procedure-name | type-name | VALUE ( expression ) }
  [ ATTR-SPACE [ = logical-expression ] ]
  [ SAVE [ = logical-expression ]
     [ INTO { directory | VALUE ( expression ) } ]
  ]
  [ LISTING { listfile | VALUE ( expression ) }
     [     APPEND [ = logical-expression ]
        |  PAGE-SIZE integer-expression
        |  PAGE-WIDTH integer-expression
     ]
  ]
  [ XCODE expression ]
  [ XREF { xreffile | VALUE ( expression ) }
     [ APPEND [ = logical-expression ] ]
  ]
  [ STRING-XREF { sxreffile | VALUE ( expression ) }
     [ APPEND [ = logical-expression ] ]
  ]
  [ STREAM-IO [ = logical-expression ] ]
  [ LANGUAGES ( { language-list | VALUE ( expression ) } )
     [ TEXT-SEG-GROW = growth-factor ] ]
  [ DEBUG-LIST { debugfile | VALUE ( expression ) } ]
  [ PREPROCESS { preprocessfile | VALUE ( expression ) } ]
  [ NO-ERROR ]
  [ V6FRAME [ = logical-expression ]
     [ USE-REVVIDEO | USE-UNDERLINE ] ]
  [ MIN-SIZE [ = logical-expression ] ]
[ GENERATE-MD5 [ = logical-expression ] ]
```

```
COMPILE { procedure-name | type-name | VALUE ( expression ) }
```

Specifies the name of the procedure file or class definition file you want to compile. Specify a class type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

If you use the SAVE option, a procedure file name must have a `.p` extension, `.w` extension, or no extension; a class definition file must have a `.cls` extension. On UNIX, file names are case sensitive, so you must enter them exactly as they are stored.

ATTR-SPACE [= *logical-expression*]

Has no effect; supported only for backward compatibility.

XCODE *expression*

Decrypts the source code in *procedure-name* or *class-name*, and any encrypted include files, using the decryption key *expression*. Only use XCODE to decrypt files not encrypted with the default key. Include files that are not encrypted are included and compiled in the standard manner.

Having the decryption key does not allow you to examine a decrypted version of the source code.

Note: You cannot use the XCODE and LISTING or DEBUG-LIST options together. Also, you cannot use the XCODE and XREF options together. That is, you cannot create a cross-reference listing from code that is encrypted.

STREAM-IO [= *logical-expression*]

Specifies that all output from the compiled procedure or class is formatted for output to a file or printer. This means that all font specifications are ignored and all frames are treated as if they had the USE-TEXT option given. This produces a platform-independent output appropriate for printing.

If you specify a *logical-expression*, its value determines whether the STREAM-IO option is activated. If the *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

```
SAVE [ = logical-expression ] [ INTO { directory | VALUE ( expression) } ]
```

Produces a file that contains the r-code for the procedure or class you are compiling.

When you compile a class definition file with the SAVE option, Progress produces an r-code file for the class definition file and all class files in its inherited class hierarchy. For example, if you compile a class definition file that has two classes in its inherited class hierarchy, Progress compiles three files and produces three r-code files.

These r-code files are saved across OpenEdge sessions. If you do not use the SAVE phrase, the COMPILE statement produces r-code for the source procedure or class, but the r-code is not saved across OpenEdge sessions. This r-code is a session-compile version of the procedure or class.

If you specify a *logical-expression*, its value determines whether the SAVE option is activated. If the *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

The COMPILE SAVE statement produces r-code files with the name *procedure-name*.r or *class-name*.r, where *procedure-name* is the name of a procedure source file without the extension, and *class-name* is the name of a class source file without the extension. Progress ignores the file extension of a procedure or class definition file and always creates r-code files that use the same filename with a .r extension. For example, if you supply a filename of test, test.p, or test.cls, COMPILE SAVE produces an r-code file with the name test.r. If you specify a filename of test.bp, COMPILE SAVE still produces an r-code file with the name test.r.

Caution: Where both procedure and class definition files compile to a .r file, be sure to use distinct filenames. If you have a procedure file and a class definition file with the same name, and you compile them both with COMPILE SAVE, the first .r file will be overwritten by the second .r file.

By default, r-code files are stored in the same directory as the source procedure or class definition file. The r-code files for inherited class definition files are also stored in the same directory as their respective source files.

If you use the SAVE INTO phrase, r-code files produced by a compilation can be saved in a different directory. See the Examples and Notes sections for more information.

On UNIX and Windows, a newly created r-code file replaces any existing r-code file of the same name.

LISTING { *listfile* | VALUE (*expression*) }

Produces a compilation listing that includes:

- The name of the file containing the procedure or class you compile.
- The date and time at the start of the compilation.
- The number of each line in the procedure or class file.
- The block number where each statement belongs.
- The complete text of all include files (except encrypted include files) and the names of any subprocedures and user-defined functions.

The *listfile* or VALUE (*expression*) identifies the name of the file in which you want to store the Compiler listing. If *expression* evaluates to the Unknown value (?), then Progress ignores the LISTING option.

APPEND [= *logical-expression*]

Appends the current listing to the contents of the listing file. If you do not use the APPEND option, Progress creates a new listing file, replacing any file of the same name.

If you specify a *logical-expression*, its value determines whether the APPEND option is activated. If the *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

PAGE-SIZE *integer-expression*

Identifies the number of lines to a page in the listing file. The default page size is 55 and *integer-expression* must be between 10 and 127, inclusive.

PAGE-WIDTH *integer-expression*

Identifies the number of page columns in the listing file. The default page width is 80, and *integer-expression* must be between 80 and 255, inclusive. Add at least 12 spaces to the page width when you type the file. This allows you to list information that precedes each line of code, ensuring that the file appears in the listing output exactly as you typed it.

```
XREF { xreffile | VALUE ( expression ) } [ APPEND [ = logical-expression ] ]
```

Writes cross-reference information between procedures and Progress objects, or between class or interface definition files and Progress objects, to the file *xreffile* or VALUE (*expression*). If *expression* returns the Unknown value (?), then Progress ignores the XREF option.

Note: You cannot use the XREF and XCODE options together. That is, you cannot create a cross-reference listing from code that is encrypted.

Cross-referenced objects include procedure and include files, user-defined functions, classes, methods, tables, fields, variables, frames, and character strings. XREF generates one unformatted, blank-separated line in *xreffile* for each referenced object. Each line has the following format:

```
source-name  file-name  line-number  reference-type  object-identifier
```

The *source-name* is the name of the procedure or class file you compile with the COMPILE XREF statement. The *file-name* is the name of the file with the referenced code. The *line-number* is the line number of the statement in *file-name* that contains the referenced object. The *reference-type* is the type of reference in the code (such as ACCESS or UPDATE), and the *object-identifier* is the Progress object being referenced.

Note: If *file-name* is an include file, *source-name* is the file that includes the include file.

The possible reference types and object identifiers appear in Table 15.

Table 15: Reference types and object identifiers *(1 of 3)*

Reference type	Object identifier
ACCESS	{ [*database.*] *table field* [WORKTABLE] } | { SHARED *variable*} | { PUBLIC-DATA-MEMBER *class-name*:*data-member-name*} | { INHERITED-DATA-MEMBER *class-name*:*data-member-name*}
CLASS	*class-name*, [INHERITS *inherited-class-name* [(*inherited-class-name* . . .)]], [IMPLEMENTS *interface-name* [*interface-name*] . . .], [USE-WIDGET-POOL], [FINAL]
COMPILE	*procedure* | *class-file*
CONSTRUCTOR	{ PUBLIC | PROTECTED} ,,,*constructor-name*,*void*, [*parameter1* [,*parameter2*] . . .]
CPINTERNAL	*name-of-the-code-page-that-Progress-uses-in-memory*
CPSTREAM	*name-of-the-code-page-that-Progress-uses-for-stream-I/O*
CREATE	[*database.*] *table* [WORKTABLE]
DATA-MEMBER	{ PUBLIC | PROTECTED | PRIVATE } *data-member-name*
DELETE	[*database.*] *table* [WORKTABLE]
DESTRUCTOR	PUBLIC,,,*destructor-name*,*void*,
DLL-ENTRY	*procedure-name*,, [*parameter1* [,*parameter2*] . . .]

Table 15: Reference types and object identifiers *(2 of 3)*

Reference type	Object identifier
EXTERN	*function-name*,*return-type*,[*parameter1* [,*parameter2*] . . .]
FUNCTION	*function-name*,*return-type*,[*parameter1* [,*parameter2*] . . .]
GLOBAL-VARIABLE	*global-variable*
INCLUDE	*include-file-name*
INTERFACE	*interface-name*,,,,
INVOKE	*class-name*:*method-name* ([*parameter-type*[,*parameter-type*] . . .])
METHOD	{ PUBLIC | PROTECTED | PRIVATE }, [OVERRIDE],[FINAL],*method-name*, *return-type*,[*parameter1*[,*parameter2*] . . .]
NEW-SHR-FRAME	*new-shared-frame*
NEW-SHR-VARIABLE	*new-shared-variable*
NEW-SHR-WORKTABLE	*new-shared-worktable* [LIKE [*database.*] *table*]
PROCEDURE	*procedure-name*,,[*parameter1* [,*parameter2*] . . .]
PUBLISH	*event-name* | (*exp*)
REFERENCE	{[*database.*] *table field* [WORKTABLE]} | {SHARED *variable*}
RUN	*procedure-name* | *value(exp)*
SEARCH	[*database.*] *table* {*index* | RECID | WORKTABLE | TEMPTABLE} [WHOLE-INDEX][1]

Table 15: Reference types and object identifiers *(3 of 3)*

Reference type	Object identifier
SHR-FRAME	*shared-frame*
SHR-WORKTABLE	*shared-worktable* [LIKE [*database.*] *table*]
SORT-ACCESS	{ [*database.*] *table field* [WORKTABLE | TEMPTABLE] }
SORT-BY-EXP	{ FOR EACH | OPEN QUERY } *table* BY *expression*
STRING	*char-string max-length justification translatable* [FORMAT]
SUBSCRIBE	*event-name* | (*exp*)
UNSUBSCRIBE	*event-name* | (*exp*) | ALL
UPDATE	{ [*database.*] *table field* [WORKTABLE] } | {SHARED *variable*} | {PUBLIC-DATA-MEMBER *class-name*:*data-member-name*} | {INHERITED-DATA-MEMBER *class-name*:*data-member-name*}

[1] WHOLE-INDEX means that the selection criteria specified to search the table does not offer opportunities to use indexes that allow optimized key references (bracketed high and low values). Instead, Progress must search the entire table using available indexes (often only the primary index) to satisfy the query, hence a WHOLE-INDEX search. Thus, depending on the query, you might be able to optimize the search by adding indexes. See also Notes.

If you specify the APPEND option, the cross-reference information is appended to an existing file. The first line of cross-reference information for a procedure contains the object identifier for the COMPILE reference type. This allows you to easily find where the information for each compilation begins. If you specify a *logical-expression*, its value determines whether the APPEND option is activated. If the *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

```
STRING-XREF { sxreffile | VALUE ( expression ) }
    [ APPEND [ = logical-expression ] ]
```

Writes cross-reference string information between procedures and Progress objects, or between class definition files and Progress objects, to the file *sxreffile* or VALUE (*expression*). If *expression* evaluates to the Unknown value (?), Progress ignores the STRING-XREF option.

```
String Xref Version x.y  source-file  code-page
```

The *x.y* is a major.minor version number, where a major version change implies a formatting change that will not be backward compatible with older versions of TranManII. The *source-file* is the name of the file from which the strings are extracted. The *code-page* is the code page with which the file was written.

The line for each string appears in the following format:

```
line-number  object-name  string  max-length  string-justification
statement-type  detail-info
```

The *line-number* is the same as *line-number* in the standard XREF file. The *object-name* is the name of the object with which the string is associated. The *max-length* and *string-justification* come from the string attribute (either explicit or implicit) and reflect the attributes applied to the string as it is entered into the text segment.

The *statement-type* describes the type of statement in which the string appears. Only one statement type appears in a given string's output line. The following values are possible:

Statement type values			
ASSIGN	DEF-SUB-MENU	INSERT	PUT-SCREEN
CASE	DISPLAY	MESSAGE	REPEAT
CREATE	DO	OPEN-QUERY	RUN
DEF-BROWSE	ENABLE	OTHER	SET
DEF-BUTTON	EXPORT	PAUSE	STATUS
DEF-FRAME	FOR	PROMPT-FOR	UPDATE
DEF-IMAGE	FORM	PUT	VIEW-AS
DEF-MENU	IF		

Note: Any statement type that is not included in the preceding list will appear as OTHER.

The *detail-info* is one or more detail tags that specify more specifically where the string appears in the statement. The following values are possible:

Detail tags			
ASSIGN	FORMAT	MESSAGE	TITLE
COL-LABEL	IMAGE-FILE	NON-ALPHA	VALUE
COMBO-BOX-ITEM	INPUT	PROMSGS	WHEN
CUR-LANG	INPUT-PARAM	PROPATH	WHERE
DEFAULT	LABEL	SEL-LIST-ITEM	WHILE
EXPR	LIST-ITEM	TERMCAP	

Note: The NON-ALPHA tag indicates that a string consists entirely of blanks or digits. The FORMAT tag is followed by one of the following tags: CHAR, NUMERIC (includes decimal and integer), DATE, or BOOL. These tags indicate the type of format. When a string can appear in only one place in a statement, no detail tag appears.

Table 16 shows the valid combinations of statement types and detail tags.

Table 16: Valid statement type and detail tag combination *(1 of 2)*

Statement type	Detail tags
ASSIGN	CUR-LANG, PROMSGS, PROPATH, TERMCAP
CASE	WHEN
CREATE	N/A
DEF-BROWSE	FORMAT, COL-LABEL
DE-FBUTTON	IMAGE-FILE, LABEL
DEF-FRAME	FORMAT, COL-LABEL, LABEL
DEF-IMAGE	IMAGE-FILE

Table 16: Valid statement type and detail tag combination *(2 of 2)*

Statement type	Detail tags
DEF-MENU	TITLE, LABEL
DEF-SUB-MENU	LABEL
DISPLAY	FORMAT, LABEL, COL-LABEL, WHEN, TITLE
DO	WHILE, WHERE, TITLE
ENABLE	LABEL, COL-LABEL, WHEN, TITLE
EXPORT	FORMAT
FOR	WHILE, WHERE, TITLE
FORM	FORMAT
IF	N/A
INSERT	TITLE
MESSAGE	TITLE, FORMAT
PAUSE	MESSAGE
PROMPT-FOR	WHEN, TITLE, FORMAT, LABEL, COL-LABEL
PUT	N/A
PUT-SCREEN	N/A
REPEAT	WHILE, TITLE, WHERE
RUN	INPUT-PARAM
SET	WHEN, ASSIGN, FORMAT, LABEL, COL-LABEL, TITLE
STATUS	DEFAULT, INPUT
UPDATE	WHEN, ASSIGN, FORMAT, LABEL, COL-LABEL, TITLE
VIEW-AS	SEL-LIST-ITEM, COMBO-BOX-ITEM

```
LANGUAGES ( { language-list | VALUE ( expression ) } )
```

Identifies which language segments to include in the compiled r-code. The *language-list* is a colon-separated list of language names used to generate each text segment. If you specify VALUE (*expression*), the expression must evaluate to a comma-separated list of language names. If *expression* evaluates to the Unknown value (?), then Progress ignores the LANGUAGES option.

Translated character strings for each specified language are read from the translation database and are stored in segments within the r-code.

```
COMPILE myfile.p LANGUAGES(French-Canadian:French:English,
                Portuguese:Spanish,
                New-York:American:English).
```

If you use an expression to specify *language-list*, you must use the VALUE option.

```
COMPILE myfile.p LANGUAGES (VALUE(char-var)).

/* char-var = "French-Canadian:French:English,
          Portuguese:Spanish,
          New-York:American:English" */
```

In this example, the compiler searches the translation database for French-Canadian translations. If a French-Canadian translation is not found, the compiler searches for a French translation. If a French translation is not found, the compiler searches for an English translation. If an English translation is not found, the compiler uses the strings from the source code.

This example generates four text segments: French-Canadian, Portuguese, New-York, and the unnamed (default) text segment. The first language name in each *language-list* argument designates the name of the text segment and specifies the first language that the compiler looks up in the translation database. As a result, it is possible to create a text segment whose name has no relationship to the languages it is composed of. For example, the following argument creates a text segment named BABEL:

```
LANGUAGES(BABEL:French:Spanish:Italian:German)
```

Provided there is no language named BABEL in the translation database, the strings in this text segment would be either French, Spanish, Italian, or German, depending on which strings have translations in which languages.

TEXT-SEG-GROW = *growth-factor*

Specifies the factor by which Progress increases the length of strings. When you develop an application that is going to be translated, it is important to allow for the growth of the text in your widgets. If you use the TEXT-SEG-GROW option, Progress increases the size of the text strings when it compiles your application.

Progress uses the following formula to determine the length of strings:

```
New-length =
Actual-length * [ 1 + (growth-factor/100 * (table-value/100)) ]
```

Where:

- *New-length* is the new string length.
- *Actual-length* is the actual string length.
- *growth-factor* is the value specified with the TEXT-SEG-GROW option.
- *table-value* is the appropriate percentage from the following table:

String length	Expansion percentage
1–10 characters	200%
11–20 characters	100%

String length	Expansion percentage
21–30 characters	80%
31–50 characters	60%
51–70 characters	40%
More than 70 characters	30%

For example, if you have a text string that is 25 characters and you specify a *growth-factor* of 50, Progress applies the formula as follows and defines the *New-length* as 35:

```
New-length = 25 * [ 1 + (80/100 * (50/100)) ]
```

Note: TEXT-SEG-GROW is supported only when you also use the LANGUAGES option.

DEBUG-LIST { *debugfile* | VALUE (*expression*) }

Writes the debug listing to the file *debugfile* or VALUE (*expression*). If *expression* evaluates to the Unknown value (?), then Progress ignores the DEBUG-LIST option. The *debugfile* consists of a line-numbered listing of the procedure with the text of all preprocessor include files, names, and parameters inserted.

PREPROCESS { *preprocessfile* | VALUE (*expression*) }

Preprocesses the procedure or class definition file and writes the preprocessed source code to the file *preprocessfile* or VALUE (*expression*). If *expression* evaluates to the Unknown value (?), Progress ignores the PREPROCESS option. The *preprocessfile* is a text file that contains a final version of your source code after all include files have been inserted and all text substitutions have been performed.

NO-ERROR

Specifies that any errors that occur as a result of the compilation are suppressed. After the COMPILE statement completes, you can check the ERROR and WARNING attributes of the COMPILER system handle to determine whether an error has occurred or any warning messages were produced. You then can check the ERROR-STATUS handle for the specific messages.

V6FRAME [= *logical-expression*] [USE-REVVIDEO | USE-UNDERLINE]

The V6FRAME option is designed specifically to compile and run Progress Version 6 applications with Progress Version 7 or later in Windows. This option uses the V6FontNumber setting in the [Startup] section of the current environment (which might be the Registry or an initialization file) to calculate the height and width of a character unit and then set the layout grid used to compile frames for display in Progress Version 7 or later.

At run time, the FONT attribute for a frame compiled with the V6FRAME option is set to the font number specified with the V6FontNumber setting. The default setting for the V6FontNumber setting is 3.

By default, V6FRAME displays a border around a fill-in field. This means that your code requires more space on the screen than in Progress Version 6. You can override this behavior with one of the following options:

- USE-REVVIDEO displays no border around a fill-in field. When a fill-in is enabled for input, the color of the fill-in changes to the color specified with the INPUT setting in the [Colors] section in the current environment (which might be the registry or an initialization file). The IBEAM cursor signals that a fill-in field has input focus.
- USE-UNDERLINE displays no border around a fill-in widget. When a fill-in is enabled for input, the underline attribute of the font (V6FontNumber) for the fill-in is turned on. The color of a fill-in enabled for input does not change. The IBEAM cursor signals that a fill-in field has input focus.

The V6FRAME option also limits the vertical size of a frame title to one character unit based upon the layout grid. The text of the frame title is in the font specified with the V6FontNumber setting in the [Startup] section of the current environment (which might be the registry or an initialization file).

The V6FRAME option governs the appearance of screen output only. Use the STREAM-IO option to compile procedures that output to files and printers. If you specify the V6FRAME and STREAM-IO options in the same COMPILE statement, the STREAM-IO option overrides the V6FRAME option.

If you specify a *logical-expression*, its value determines whether the V6 compile option is activated. If the *logical-expression* is evaluated to the Unknown value (?), a run-time error occurs.

For more information on the environment for an OpenEdge session, see *OpenEdge Deployment: Managing 4GL Applications*.

MIN-SIZE [= *logical-expression*]

Minimizes the size of the generated r-code file by eliminating the Debugger Segment (which is used by the OpenEdge Debugger) and the signature descriptor data (which is used by the Open Client Proxy Generator).

If you specify a *logical-expression*, its value determines whether the MIN-SIZE option is activated (TRUE) or not (FALSE). If the *logical-expression* evaluates to the Unknown value (?), a run-time error occurs. The default value is FALSE.

GENERATE-MD5 [= *logical-expression*]

When Progress compiles a procedure or class definition file with the GENERATE-MD5 option, it generates a special MD5 value based on the code content, and stores it in the r-code file. This r-code MD5 value is similar to a CRC value, except the MD5 value is 128 bits in size and the CRC value is only 16 bits. The MD5 value is virtually guaranteed to be different if the file content has changed. As with CRC, content changes include any schema changes. That is, if only the schema changes, the MD5 value also changes.

If you specify a *logical-expression*, its value determines whether the GENERATE-MD5 option is activated (TRUE) or not (FALSE). The default value is TRUE.

You can read the MD5-VALUE attribute on the RCODE-INFO system handle to determine the MD5 value for a procedure or class.

This option is supported for WebClient™ only (that is, only WebClient uses the resulting MD5 value). Progress recommends compiling your WebClient application procedures with this option. Using this option lets WebClient determine if an r-code file has changed since the previous version of the application.

Examples

In this procedure, Progress creates an r-code version of the `ord-ent` procedure, naming it `ord-ent.r`:

r-cmple.p

```
COMPILE ord-ent SAVE.
```

Note: The sample procedures supplied with Progress do not include the `ord-ent` procedure.

In this procedure, Progress compiles the demo1 procedure, reserving spaces in frame layouts for special field attributes and producing an r-code file, `demo1.r`, that can be used across OpenEdge sessions. Progress saves the r-code file in the current directory.

r-cmple2.p

```
COMPILE demo1 ATTR-SPACE SAVE.
```

You can save the r-code file in a different directory by using the SAVE INTO phrase. For example, to save an r-code file in `/usr/sources` on a UNIX system, enter this command:

```
COMPILE demo1 ATTR-SPACE SAVE INTO /usr/sources.
```

The following example shows the effect of include files on compilation listings:

r-incl.p

```
FOR EACH customer:
  {r-fcust.i}
  {r-dcust.i}
END.
```

Suppose you use the following COMPILE statement to compile the `r-incl.p` procedure:

r-comlis.p

```
COMPILE r-incl.p  SAVE  LISTING r-incl.lis
 XREF r-incl.xrf DEBUG-LIST r-incl.dbg.
```

This COMPILE statement produces four files: `r-incl.r`, `r-incl.lis`, `r-incl.xrf`, and `r-incl.dbg`.

The following procedures contain the contents of the `r-incl.lis`, `r-incl.xrf`, and `r-incl.dbg` files:

r-incl.lis

```
r-incl.p                        06/01/93 13:06:30   PROGRESS(R) Page 1

{} Line Blk
-- ---- ---
      1     /* r-incl.p */
      2
      3   1 FOR EACH customer:
      4   1   {r-fcust.i}
 1    1   1 /* r-fcust.i */
 1    2   1
 1    3   1 FORM customer.cust-num customer.name LABEL "Customer Name"
 1    4   1      customer.phone FORMAT "999-999-9999".
      4   1
      5   1   {r-dcust.i}
 1    1   1 /* r-dcust.i */
 1    2   1
 1    3   1 DISPLAY customer.cust-num customer.name customer.phone.
      5   1
      6     END.
^Lr-incl.p                      06/01/93 13:06:30   PROGRESS(R) Page 2

     File Name       Line Blk. Type Tran            Blk. Label
-------------------- ---- --------- ---- --------------------------------
r-incl.p                 0 Procedure No
r-incl.p                 3 For       No
    Buffers: sports.Customer
    Frames:  Unnamed

^L
```

This sample output is not an exact copy of the `r-incl.lis` file.

There are three columns next to the procedure in the listing file:

1. **{}** — The level of the include file.
2. **Line** — The line number in the file.
3. **Blk** — The number of the block.

The information follows each of the procedure blocks or function blocks:

- **Line** — The line number where the block starts.
- **Blk. Type** — The type of block (Procedure, DO, FOR EACH, REPEAT).
- **Tran** — Whether the block is a transaction block.
- **Blk. Label** — The label of the block.
- **Buffers** — The name of the record buffer scoped to the block.
- **Frames** — The name of the frame scoped to the block.

This is the cross-reference file `r-incl.xrf`:

r-incl.xrf

```
r-incl.p r-incl.p 1 COMPILE r-incl.p
r-incl.p r-incl.p 3 STRING "Customer" 8 NONE UNTRANSLATABLE
r-incl.p r-incl.p 3 SEARCH sports.Customer Cust-Num
r-incl.p r-incl.p 4 INCLUDE r-fcust.i
r-incl.p r-fcust.i 3 ACCESS sports.Customer Cust-Num
r-incl.p r-fcust.i 3 ACCESS sports.Customer Name
r-incl.p r-fcust.i 3 ACCESS sports.Customer Phone
r-incl.p r-fcust.i 3 STRING ">>>>9" 5 NONE TRANSLATABLE  FORMAT
r-incl.p r-fcust.i 3 STRING "x(20)" 5 NONE TRANSLATABLE  FORMAT
r-incl.p r-fcust.i 3 STRING "999-999-9999" 12 NONE TRANSLATABLE  FORMAT
r-incl.p r-incl.p 5 INCLUDE r-dcust.i
r-incl.p r-dcust.i 3 ACCESS sports.Customer Cust-Num
r-incl.p r-dcust.i 3 ACCESS sports.Customer Name
r-incl.p r-dcust.i 3 ACCESS sports.Customer Phone
r-incl.p r-incl.p 6 STRING "Cust-Num" 8 LEFT TRANSLATABLE
r-incl.p r-incl.p 6 STRING "Customer Name" 13 LEFT TRANSLATABLE
r-incl.p r-incl.p 6 STRING "Phone" 5 LEFT TRANSLATABLE
r-incl.p r-incl.p 6 STRING
    "-------- ---------------------- --------------" 46 LEFT TRANSLATABLE
r-incl.p r-incl.p 6 STRING "Cust-Num" 8 LEFT TRANSLATABLE
```

Each line in the `xref` file specifies the procedure, line number, access type, and access information. The first line in the `xref` file contains the COMPILE access type directive and the name of the procedure exactly as it appears in the COMPILE statement. See Table 15 for a list of the values that follow a particular access type (for example, *`table`* and *`index`* after SEARCH).

This is the debug listing `r-incl.dbg`:

r-incl.dbg

```
 1   /* r-incl.p */
 2
 3   FOR EACH customer:
 4
 5   /* r-fcust.i */
 6
 7   FORM customer.cust-num customer.name LABEL "Customer
 8         Name" customer.phone FORMAT "999-999-9999".
 9
10
11   /* r-dcust.i */
12
13  DISPLAY customer.cust-num customer.name customer.phone
14
15   END.
```

Notes

- When compiling class definition files, the following options apply to the class definition file identified in the COMPILE statement and all class files in its inherited class hierarchy: XCODE, STREAM-IO, LANGUAGES, V6FRAME, MIN-SIZE and GERNERATE-MD5.
- When compiling class definition files, the following options apply only to the class definition file identified in the COMPILE statement, and not to the class files in its inherited class hierarchy: PREPROCESS, LISTING, DEBUG-LIST, XREF, and STRING-XREF.
- If you want all record retrieval statements in a procedure to default to NO-LOCK, you must compile the procedure in an OpenEdge session started with the No Lock (-NL) startup parameter. For more information on record locking, see *OpenEdge Development: Progress 4GL Handbook*. For more information on the No Lock (-NL) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- The value of the `PROPATH` environment variable defines the list of directories (path) to use when searching for a procedure.

- On UNIX, you define the PROPATH variable in a startup script or in your .profile file. In Windows, you can define your PROPATH in the Registry or in an initialization file. You can also define the PROPATH interactively at the operating system level.

  In addition to any directories you define for PROPATH, Progress searches the directory containing the Progress system software. If you do not define a value for PROPATH, Progress searches your working directory by default.

- To locate the source file that you name in the COMPILE SAVE statement, Progress searches the first directory in PROPATH. If the source file is there, Progress compiles the source file and creates an r-code file. On UNIX, this new r-code file replaces any existing r-code file. If errors occur during compilation, Progress does not produce an r-code file and leaves existing r-code files unchanged.

  If Progress cannot find the source file, it continues on to the next directory in PROPATH.

- Use the SAVE INTO phrase to store a compiled r-code file in a different directory from its corresponding source file.

  If you specify a relative pathname for the source file, that pathname is appended to the SAVE INTO path. For example (using UNIX pathnames):

```
PROPATH="/pro1/source".
COMPILE  test/proc1.p SAVE INTO /pro1/obj.
```

  In the example, Progress saves the source file /pro1/source/test/proc1.p as /pro1/obj/test/proc1.r.

  If the source file is a full pathname, Progress stores the r-code file in the SAVE INTO directory; it drops its original directory path.

```
COMPILE /pro1/obj/test/proc1.p SAVE INTO /usr/rcode.
```

  In the example, Progress saves the source file as /usr/rcode/proc1.r.

  When you use the SAVE INTO phrase to store compiled r-code files for one or more class definition files specified with a package, Progress creates a directory structure under the specified SAVE INTO directory that is consistent with the directory structure of the original source files relative to PROPATH (if the directory structure doesn't already exist). That is, Progress creates a subdirectory under the specified SAVE INTO directory to match the original source directory for each class definition file in the hierarchy.

For example, if the source for two class definition files in a class hierarchy reside in two different directories, such as `dir1` and `dir2`, Progress creates two matching subdirectories named `dir1` and `dir2` under the specified SAVE INTO directory and stores the r-code files in their respective subdirectories.

If the SAVE INTO pathname is null, Progress saves r-code files in the same directory as their source files.

- The ATTR-SPACE/NO-ATTR-SPACE designation in a Frame phrase takes precedence over an ATTR-SPACE/NO-ATTR-SPACE designation in a Format phrase. The ATTR-SPACE/NO-ATTR-SPACE designation in a Format phrase takes precedence over an ATTR-SPACE/NO-ATTR-SPACE designation in a COMPILE statement.
- To locate a file with the COMPILE statement (without the SAVE phrase), Progress searches the first directory in `PROPATH` for a usable r-code file. A usable r-code file must meet these criteria:
  - It must have the correct format; it must have been produced by the COMPILE SAVE statement.
  - It must have been produced by the current version of the Progress Compiler.
  - It must have the same cyclic redundancy check (CRC) value as any database tables it references, or the same time stamp if you are running with the Timestamp (-tstamp) parameter. When creating an r-code file, Progress includes, as part of the r-code file, either the CRC or the time stamp of the most recent change to the database schema that affects **this** procedure (for example, adding or deleting a field or index definition in a table that the procedure references).
  - On UNIX, it must have read access to the r-code file.

  If there is a usable r-code file, there is no reason to perform the compilation. You receive an error and the compilation stops unless you have specified the XREF, LISTING, PREPROCESS, or DEBUG-LIST option. If you specified one of these options, Progress continues with the compilation and produces the files specified and a session compile. If Progress does create a session compile version, the version is not used when you use the RUN statement. The RUN statement always uses an existing r-code file before using a session compile version of a procedure.

  If there is no usable r-code file, Progress searches the same directory in `PROPATH` for a source file. If the source file is there, Progress compiles it into the session compile file. If it is not there, Progress continues on to the next directory in `PROPATH`, searching for an r-code file, then for a source file.

- After you compile a procedure, the RUN statement does not recompile it. If you RUN a procedure multiple times within a session, changing the procedure between runs, you must manually recompile the procedure each time. Otherwise, the procedure's last r-code, which persists for a session, is found and the procedure is not automatically recompiled.
- The size of the r-code might vary, depending on the window system on which it is compiled.
- Modifications to existing field definitions do not affect database table CRC or time-stamp values. Therefore, updating a table's existing field definitions does not invalidate r-code versions of procedures that reference the table. However, adding or deleting tables, fields, or indexes does affect database table CRC and time stamps. This invalidates r-code versions of procedures that reference the changed tables.
- When you use a reserved keyword to specify a language with the LANGUAGES option, you must use quotation marks (" ") around the *language-list*.
- The SORT-BY-EXP reference in the XREF is used to indicate a FOR EACH or OPEN QUERY statement which contains a BY clause which uses an expression.

- A WHOLE-INDEX search reported for a table occurs when an entire index is used to search the table. (That is, the bracket used by the query to search the table spans the entire index.) This can occur either when no selection criteria are specified to limit the range of index keys searched (that is, to bracket a subset of the index) or when there is no appropriate index available to optimize the selection criteria. For example, the following queries on Customer table of the sports database both result in WHOLE-INDEX searches. The first query uses the Name index to search the entire table, returning every record in Name order. The second query uses the primary index to search the entire table because there is no index provided for the Balance field to limit the search.

```
FOR EACH Customer USE-INDEX Name:
    DISPLAY Customer.
END.

FOR EACH Customer WHERE Balance < 10000 AND Balance > 5000:
    DISPLAY Customer.
END.
```

  On the other hand, the following queries do not result in WHOLE-INDEX searches because the selection criteria directly limit the range of Name and Cust-Num index keys (respectively) to be searched:

```
FOR EACH Customer
    WHERE Name < "Penan Sporttiklubi" AND Name > "Chip's Poker":
    DISPLAY Customer.
END.

FOR EACH Customer WHERE Cust-Num < 40:
    DISPLAY Customer.
END.
```

- For SpeedScript, the following options are invalid: V6FRAME, USE-REVVIDEO, and USE-UNDERLINE.

See also COMPILER system handle, NEW statement, RUN statement, No Lock (-NL) Startup Parameter (in *OpenEdge Deployment: Startup Command and Parameter Reference*)

CONNECT statement

Allows access to one or more databases from within a Progress procedure.

Syntax

```
CONNECT
  {
    { physical-name | VALUE ( expression ) } [ options ] | options
  }
  [ NO-ERROR ]
```

physical-name

The actual name of the database on disk. It can be a simple filename, relative pathname, or a fully qualified pathname, represented as an unquoted string, a quoted string, or a VALUE (*expression*). If you do not give a fully qualified pathname, Progress searches for the database relative to your current directory.

VALUE (*expression*)

An expression (a constant, field name, variable name, or expression) whose value is a string representing the physical name of a database to connect.

options

One or more options, similar to those used to start Progress. Valid options are a subset of OpenEdge startup parameters. Note that parameters are case sensitive.

See *OpenEdge Deployment: Startup Command and Parameter Reference* for more information on OpenEdge startup parameters.

NO-ERROR

Suppresses errors in the act of connecting. This does not mean that all errors produced by the server are suppressed; only errors caused by the CONNECT statement itself. For example, if the server to which you are connecting runs out of resources, its error message will not be suppressed. If a CONNECT error occurs (for example, the database does not exist or is in use in single-user mode), error information is written to the ERROR-STATUS system handle.

You also can use the CONNECTED function to determine whether the CONNECT succeeded and then retrieve error messages from the ERROR-STATUS handle.

Examples

This procedure attempts to connect to databases `mydb1` and `mydb2` in single-user mode, with error suppression. You must connect to a database before you run a procedure that references it.

r-connct.p

```
CONNECT mydb1 -1 -db mydb2 -1 NO-ERROR.
```

In the next example, assume database `sports` has not been previously connected, so the following `r-cnct1.p` procedure fails. At the start of execution, `r-cnct1.p` checks whether `sports` is connected. If `sports` is not connected, a run-time error occurs. As shown in the example, attempting to connect to `sports` within the procedure does not solve the problem.

```
/* NOTE: this code does NOT work */

CONNECT sports -1.
FOR EACH sports.customer:
  DISPLAY customer.
END.
```

Instead, split r-cnct1.p into two procedures, as shown in r-dispcu.p and r-cnct2.p:

r-dispcu.p

```
FOR EACH sports.customer:
    DISPLAY customer.
END.
```

r-cnct2.p

```
CONNECT sports -1.
RUN "r-dispcu.p".
```

This time, database sports is connected before r-dispcu.p is invoked, so r-dispcu.p runs successfully.

Notes

- Each connected database is assigned a logical name for the current session, and is referred to by this logical name during the session. Use the Logical Database Name (-ld) parameter to specify a logical name. If the logical name is not specified using the -ld parameter, then the physical database filename, without the .db suffix, is the default logical name. For example, if the physical name is /users/eastcoast/proapp/mydb.db, then the default logical name is mydb. Logical names are **not** case sensitive.
- Databases can have aliases (see also ALIAS function). A database can have more than one alias, but each alias refers to only one database. The first database connected during a given session automatically receives the alias DICTDB. The first database connected that has a _menu file automatically receives the alias FTDB. You can reassign the FTDB alias to any other FAST TRACK database.
- When you try to connect the same database twice using the same logical name, Progress returns a warning, which you can suppress with NO-ERROR.
- When you try to connect different databases using the same logical name, Progress returns an error message and an error condition. You can suppress the error condition with NO-ERROR, and test with the CONNECTED function.

- When you try to connect to multiple databases and a connection fails, a run-time error occurs. The successfully connected databases remain connected and program execution continues. Use the CONNECTED function to find out which databases are successfully connected.

- If you run a procedure that requires a database and that database is not connected, OpenEdge searches for the database in the auto-connect lists in all connected databases. If OpenEdge finds the required database there, it automatically attempts to connect to the database with the parameters set for it in the auto-connect list. You can edit the auto-connect list using the database utilities in the OpenEdge Data Dictionary. If OpenEdge does not find it, the connection attempt fails.

- Connection information found in an OpenEdge auto-connect list is merged with connection information in a CONNECT statement that connects the database. So, if you connect a database with a CONNECT statement, and that database already has an entry in the OpenEdge auto-connect list of a connected database, the connection information in the auto-connect list and the CONNECT statement is merged. However, the connection information in the CONNECT statement takes precedence.

- Permission issues limit the use of the CONNECT statement for raw I/O connections to databases in single-user and multi-user direct-access mode on UNIX machines that do not support O_SYNC and SWRITE.

  At startup, the OpenEdge client executable has superuser privileges that allow it to open raw disk devices. Thus, you can open any databases specified on the startup command line with raw I/O. After startup, the client executable relinquishes the superuser privileges that allow it to open raw disk devices. As a result, you cannot use the CONNECT statement to establish a raw I/O connection to a database in single-user or multi-user direct-access mode.

  When you try to use a CONNECT statement to open a raw I/O connection to a database in single-user mode, OpenEdge establishes a buffered (non-raw) I/O connection to the database and displays a non-raw warning message.

- When you try to use a CONNECT statement to open a raw I/O connection to a database in multi-user direct-access mode, one of the following events occur:

  – If you started a server (PROSERVE) for the database with the Buffered I/O (-r) parameter, OpenEdge establishes a non-raw I/O connection to the database.

  – If you started a server (PROSERVE) for the database with the Raw I/O (-R) parameter, the CONNECT statement fails.

  There are several ways to avoid these problems:

  – Establish raw I/O database connections in the single-user and multi-user direct-access modes at OpenEdge startup.

  – If you must use the CONNECT statement to establish a raw I/O database connection, establish the connection with the Client Multi-user (-cl) parameter. Be sure to start the database server (PROSERVE) with the Raw I/O (-R) parameter before you do this.

  – If you must use the CONNECT statement to establish a raw I/O database connection in single-user or multi-user direct access mode on UNIX, follow these steps **carefully**:

    1. Change the permissions of the OpenEdge client executable to rwsrwsr-x by typing **chmod 6775 _progres.**

    2. Change the group of the client executable to match the group of the raw device (for example, `/dev/rsd0d`) and block special device (for example, `/dev/sd0d`).

    3. Change the permissions of the raw and block special devices to "rw-rw----".

The disadvantage of this procedure is that all files produced within OpenEdge have the same group as the disk device. Consider the following:

- If you want to run a multi-user direct-access session in non-raw mode, you must start the database server with the Buffered I/O (-r) parameter.
- If a database and accompanying before-image file have read-only permissions (r--r--r--) and you try to connect to that database in single-user or multi-user mode using the CONNECT statement, the connection will fail with the following error:

```
errno=13
```

This connection failure results because the _progres module relinquishes superuser privileges after start-up and no longer possesses the privileges required to connect to the database using the CONNECT statement.

- For more information on connecting to databases, see *OpenEdge Development: Programming Interfaces*.

See also ALIAS function, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

CONNECTED function

Tells whether a database is connected. If *logical name* is the logical name or *alias* is the alias of a connected database, the CONNECTED function returns TRUE; otherwise, it returns FALSE.

Syntax

```
CONNECTED ( logical-name | alias )
```

logical-name

Refers to a logical name. It can be a quoted string or a character expression. An unquoted character string is not allowed.

alias

Refers to an alias. It can be a quoted string or a character expression. An unquoted character string is not allowed.

Example

This procedure runs `r-dispcu.p` if a database with the logical name sports is connected:

r-cnctd.p

```
IF CONNECTED("sports") THEN RUN r-dispcu.p.
```

See also

ALIAS function, CONNECT statement, CREATE ALIAS statement,CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

CONSTRUCTOR statement

Defines a constructor method for a class. Progress invokes this constructor method to initialize data or state for a new class object instance when the object is instantiated using the NEW statement.

Note: This statement is applicable only when used in a class definition (`.cls`) file.

Syntax

```
CONSTRUCTOR { PROTECTED | PUBLIC } class-name
  ( [ parameter [, parameter ] ... ] ):

  constructor-body
```

{ PROTECTED | PUBLIC }

Specifies the access mode for this constructor method.

A class with a PROTECTED constructor method can be accessed only by an inheriting class.

A class with a PUBLIC constructor method can be accessed by the defining class, any of its inheriting classes, and any class or procedure that instantiates the class using the NEW statement.

class-name

The name of the class this method constructs. This name must match the class name portion of the type name for the class (that is, the name of the class definition file excluding the `.cls` extension and any package path information).

(*parameter* [, *parameter*] ...)

Optionally specifies one or more parameters of the constructor method.

For the parameter definition syntax, see the Parameter definition syntax reference entry in this book.

The NEW statement, which creates an object instance of the class, must provide for the parameters identified by this constructor method. The parameters must match with respect to the number, data type, and mode. For more information about the NEW statement, see the NEW statement reference entry in this book.

constructor-body

The body of the constructor definition. Define the constructor body using the following syntax:

```
      .
      .
      .
method-logic
      .
      .
      .
END [ CONSTRUCTOR ].
```

method-logic

The logic of the constructor method, which can contain any Progress 4GL statements currently allowed within a PROCEDURE block including class-related statements, but excluding the RETURN ERROR statement. This typically contains logic to initialize the data members in the class.

If the defining class of this constructor method is a subclass and its super class contains a constructor method that takes parameters, the first executable statement in this constructor method must invoke the constructor method for the super class using the SUPER( ) method and the parameters must match with respect to the number, data type, and mode. For more information, see the SUPER( ) method reference entry in this book.

If the constructor method for the super class does not take parameters, you need not invoke it. Progress automatically invokes the constructor method for the super class when it instantiates the class object.

END [CONSTRUCTOR]

Specifies the end of the constructor body definition. You must end the constructor body definition with the END statement.

Example The following example shows the definition of a constructor method:

```
CONSTRUCTOR PUBLIC CustObj( ):

  m_NumCusts = 0.
  /* Fill a temp table and get the row count */

  FOR EACH Customer NO-LOCK:
    CREATE ttCust.
    ASSIGN
      ttCust.CustNum = Customer.CustNum
      ttCust.Name = Customer.Name
      m_NumCusts = m_NumCusts + 1.
  END.

END CONSTRUCTOR.
```

Notes

- You can terminate a CONSTRUCTOR statement with either a period (.) or a colon (:).
- A constructor method definition must begin with the CONSTRUCTOR statement and end with the END statement.
- A constructor method has no return type.
- You never explicitly invoke the constructor method to create a class object instance. The method is implicitly invoked when the object is instantiated with the NEW statement, or explicitly invoked by the constructor method in an inheriting subclass using the SUPER( ) method.
- Any application logic errors that occur within the constructor method must be programmatically propagated back to the calling procedure. Consider using an output parameter to return the error condition to the calling procedure, or deleting the object from within the constructor. For more information, see *OpenEdge Getting Started: Object-oriented Programming*.

See also CLASS statement, DESTRUCTOR statement, FUNCTION statement, NEW statement, SUPER( ) method

COPY-LOB statement

Copies large object data between BLOBs, CLOBs, MEMPTRs, and LONGCHARs. It also copies large object data to and from the file system, and converts large object data to or from a specified code page.

Note: You cannot copy large object data between BLOBs and CLOBs directly. However, you can copy a BLOB or CLOB to a MEMPTR or LONGCHAR (which converts the data) and then copy the MEMPTR or LONGCHAR to the CLOB or BLOB, respectively.

Syntax

```
COPY-LOB
  [ FROM ] { [ OBJECT ] source-lob | FILE source-filename }
     [ STARTING AT n ] [ FOR length ]
  TO { [ OBJECT ] target-lob [OVERLAY AT n [TRIM ] ] |
     FILE target-filename [ APPEND ] }
  [ NO-CONVERT | CONVERT convert-phrase ]
  [ NO-ERROR ].
```

[OBJECT] *source-lob*

The source object to be copied, which can be a MEMPTR or LONGCHAR variable, a BLOB or CLOB database or temp-table field, or a dynamic expression that resolves to a BLOB or CLOB database or temp-table field. The source object data at this location is copied to the specified target object or file.

FILE *source-filename*

A character expression that specifies the name of a file containing the source object data to be copied. The object data in this source file is copied to the specified target object or file. You can specify an absolute or relative pathname.

Progress raises the ERROR condition if *source-filename* resolves to the Unknown value (?) or the source file cannot be read.

`STARTING AT` *n*

An integer expression indicating a one-based offset position, in the source object or file, from which to start copying. The copy begins at offset 1, by default. Progress raises the ERROR condition if the specified offset position is less than 1, greater than the size of the object or file, or the Unknown value (?).

Note: Offsets are measured in bytes for binary data (BLOB or MEMPTR), and characters for character data (CLOB or LONGCHAR).

`FOR` *length*

An integer expression indicating the number of bytes or characters to copy from the source object or file starting at the specified offset position. Progress copies from the specified offset position to the end of the object or file, by default. Progress raises the ERROR condition if the specified length is less than 0, greater than the size of the object or file, or the Unknown value (?).

Note: Offsets are measured in bytes for binary data (BLOB or MEMPTR), and characters for character data (CLOB or LONGCHAR).

[`OBJECT` **]** *target-lob*

The target object to receive the copy, which can be a MEMPTR or LONGCHAR variable, a BLOB or CLOB database or temp-table field, or a dynamic expression that resolves to a BLOB or CLOB database or temp-table field. The object data in the specified source object or file is copied to the target object.

If the specified target object does not yet exist, Progress either creates a BLOB or a CLOB, or allocates memory for a MEMPTR or a LONGCHAR. If the specified target object already exists, Progress deletes the object before the copy operation begins, by default. You can specify the `OVERLAY AT` *n* option to overlay some portion of an existing target object.

Note: Although Progress allocates memory for a target MEMPTR, you are responsible for freeing that memory.

`OVERLAY AT` *n* **[** `TRIM` **]**

An overlay position in the target object. Progress copies the source object or file to an existing BLOB, CLOB, MEMPTR, or LONGCHAR target starting at the given position. If the operation results in writing past the end of a target BLOB, CLOB, or LONGCHAR, Progress extends the target object as necessary. If the operation results in writing past the end of a target MEMPTR, Progress raises the ERROR condition.

If the target object does not yet exist, Progress raises the ERROR condition. If the specified overlay position is less than 1, greater than the size of the object, or the Unknown value (?), Progress raises the ERROR condition.

You can specify the TRIM option only if the target object is a BLOB or CLOB. In this case, Progress copies the source object or file to the existing target object and truncates any data remaining in the target object. If the target object is a MEMPTR or LONGCHAR, Progress ignores this option.

`FILE` *target-filename* **[** `APPEND` **]**

A character expression that specifies the name of the target file to which the object data in the specified source object or file is copied. You can specify an absolute or relative pathname.

If the target file does not exist, Progress creates the file. If the target file exists, and you specify the `APPEND` option, Progress opens the file and appends the object data to the end of a file. If the target file exists, but you do not specify the `APPEND` option, Progress creates the target file anew (which overwrites the original file).

If *target-filename* resolves to the Unknown value (?), or the target file cannot be created or written, Progress raises the ERROR condition.

`NO-CONVERT` **|** `CONVERT` *convert-phrase*

Lets you specify the character conversion behavior between the source and target objects.

The `NO-CONVERT` option specifies that no conversions occur. However, if the target is a LONGCHAR or a CLOB, Progress validates the character data based on the target object's code page. For a CLOB, this is the code page of the CLOB. For a LONGCHAR, this is `-cpinternal` unless the LONGCHAR's code page was set using the FIX-CODEPAGE function. If the validation fails, Progress raises the ERROR condition.

The CONVERT option lets you specify how Progress converts object data. Following is the syntax for *convert-phrase:*

```
{
  [ SOURCE CODEPAGE codepage ]
  [ TARGET CODEPAGE codepage ]
}
```

Specify SOURCE CODEPAGE to indicate that a source object is in the specified code page. If you specify TARGET CODEPAGE, Progress converts the target object to the specified code page.

Table 17 lists the default character conversions Progress performs when copying data between the source and target objects. References to CLOBCP and CLOBDB represent CLOB data in either the CLOB's defined code page or the database's defined code page, respectively. References to the "fixed code page" represent the code page of a target LONGCHAR variable set using the FIX-CODEPAGE function.

Table 17: Default COPY-LOB statement character conversions *(1 of 2)*

When the source object is a ...	And the target object is a ...	Progress converts the source object ...
MEMPTR	LONGCHAR	From -cpinternal to -cpinternal or the fixed code page.
MEMPTR	CLOBDB	From -cpinternal to the database's defined code page.
MEMPTR	CLOBCP	From -cpinternal to the CLOB's defined code page.
BLOB	LONGCHAR	No conversion, the LONGCHAR is in -cpinternal or the fixed code page.
LONGCHAR	MEMPTR	From the LONGCHAR's code page to -cpinternal.
LONGCHAR	BLOB	No conversion, the BLOB's code page is unknown.

Table 17: Default COPY-LOB statement character conversions *(2 of 2)*

When the source object is a ...	And the target object is a ...	Progress converts the source object ...
LONGCHAR	CLOBDB	From the LONGCHAR's code page to the database's defined code page.
LONGCHAR	CLOBCP	From the LONGCHAR's code page to the CLOB's defined code page.
CLOBDB	MEMPTR	From the database's defined code page to `-cpinternal`.
CLOBDB	LONGCHAR	From the database's defined code page to `-cpinternal` or the fixed code page.
CLOBCP	MEMPTR	From the CLOB's defined code page to `-cpinternal`.
CLOBCP	LONGCHAR	No conversion, or conversion to the fixed code page.

Note: If either the source or target object is a file, the target's code page defaults to `-cpstream`.

`NO-ERROR`

Suppresses any errors that occur as a result of the copy operation. After the statement completes, you can check the ERROR-STATUS system handle for information about any errors that might have occurred.

Notes

- If a source or target object is stored in a database, its record must be available to copy. The lock mode of the record containing the target object must be EXCLUSIVE-LOCK or SHARE-LOCK and upgradeable; otherwise, the COPY-LOB statement raises the ERROR condition.
- You can also assign large object data from one BLOB or MEMPTR to another, and one CLOB or LONGCHAR to another, using the = Assignment operator or ASSIGN statement. You cannot use the = Assignment operator or ASSIGN statement to assign large object data between BLOBs or MEMPTRs and CLOBs or LONGCHARs.

COUNT-OF function

Returns an integer that is the total number of selected records in the file or files you are using across break groups.

Syntax

```
COUNT-OF ( break-group )
```

break-group

The name of a field or expression you named in the block header with the BREAK BY option.

Example

This procedure sorts all customers by state and then calculates the percentage of the total number of customers that are in each state. The COUNT-OF function provides the calculation with the number of customer records in the database.

r-cntof.p

```
FOR EACH customer BREAK BY state:
  DISPLAY cust-num name sales-rep state.
  ACCUMULATE state (SUB-COUNT BY state).
  IF LAST-OF(state)
  THEN DISPLAY 100 * (ACCUM SUB-COUNT BY state state) / COUNT-OF(state)
    FORMAT "99.9999%"
    COLUMN-LABEL "% of Total!Customers".
END.
```

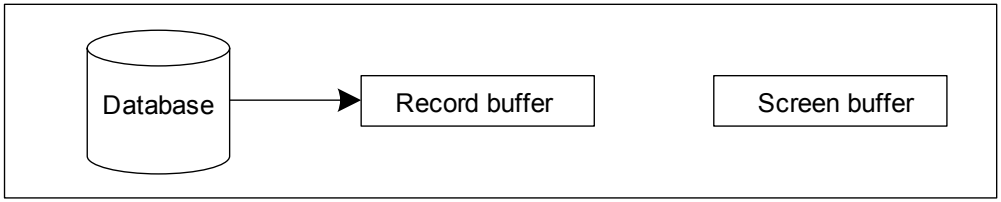
See also

Aggregate phrase

CREATE statement

Creates a record in a file, sets all the fields in the record to their default initial values, and moves a copy of the record to the record buffer.

Data movement



Syntax

```
CREATE record
  [ USING { ROWID ( nrow ) | RECID ( nrec ) } ] [ NO-ERROR ]
```

record

The name of the record or record buffer you are creating.

To create a record in a file defined for multiple databases, you might have to qualify the record's filename with the database name. See the Record phrase reference entry for more information.

`USING { ROWID ( nrow ) | RECID ( nrec ) }`

Supported only for backward compatibility.

NO-ERROR

Specifies that any errors that occur in the attempt to create the record are suppressed. After the CREATE statement completes, you can check the ERROR-STATUS system handle for information on any errors that might have occurred.

Example

The following example creates a record in the order file for each pass through the loop and then updates the record. It also creates an order-line record.

r-create.p

```
REPEAT:
    CREATE order.
    UPDATE order.order-num
           order.cust-num VALIDATE(CAN-FIND(customer OF order),
                                   "Customer does not exist")
           cust-num order-date.
    REPEAT:
       CREATE order-line.
       order-line.order-num = order.order-num.
       UPDATE line-num
       order-line.item-num VALIDATE(CAN-FIND(item OF order-line),
                                    "Item does not exist")
       qty
       price.
    END.
END.
```

This procedure adds orders and order-lines to the database. Because the user supplies an order number when updating the order record, that order number is assigned (=) to the order-num field of the order-line record when the order-line record is created.

Notes

- When you run procedures that create large numbers of records (for example, during initial data loading), the process runs much faster if you use the No Crash Protection (-i) parameter. See *OpenEdge Deployment: Startup Command and Parameter Reference* for more information on startup parameters. Back up your database before you use this parameter.
- After you create a new record with CREATE, Progress waits to write the record to the database until after the next statement generates an index entry for the record.
- The CREATE statement causes any related database CREATE triggers to execute. All CREATE triggers execute after the record is actually created. If a CREATE trigger fails (or executes a RETURN statement with the ERROR option), the record creation is undone. See *OpenEdge Development: Progress 4GL Handbook* for more information on database triggers.

See also INSERT statement, NEW function

CREATE ALIAS statement

Creates an alias for a database. Once an alias is created, it can be used in place of the database's logical name.

Note: A database can have more than one alias, but each alias refers to one and only one database.

Syntax

```
CREATE ALIAS alias-string | value ( expression )
  FOR DATABASE logical-name-string | value ( expression )
  [ NO-ERROR ]
```

alias-string | value (*expression*)

An unquoted string, quoted string, or CHARACTER expression that represents an alias for the database.

FOR DATABASE *logical-name-string* | value (*expression*)

An unquoted string, quoted string, or CHARACTER expression that represents the logical name of the database.

Note: The logical name must already be set.

NO-ERROR

Tells Progress to allow the alias to be created even if the database is not connected.

If you CREATE ALIAS for a database that is not connected and omit NO-ERROR, Progress reports a run time error.

Note: The NO-ERROR option of the CREATE ALIAS statement behaves differently from the NO-ERROR option of other 4GL elements.

Example

This procedure creates the alias `myalias` for database `mydb`:

r-cralas.p

```
CREATE ALIAS myalias FOR DATABASE mydb NO-ERROR
```

Notes

- The first OpenEdge database connected during a given session receives the DICTDB alias.
- The first database connected that has an `_menu` file automatically receives the alias FTDB. You can reassign the FTDB alias to any other `FAST TRACK` database.
- If there is already a database connected with logical name equal to *alias*, CREATE ALIAS fails.
- If there is an existing alias equal to *alias*, the existing alias is replaced by the new alias.
- If you want to use an expression for an alias name or logical name, you must use CREATE ALIAS VALUE (*expression*) FOR DATABASE VALUE (*expression*).
- When a given database is disconnected, the existing aliases that refer to it are not erased, but remain in the session alias table. Later in the same session, if you connect to a database with the same logical name, the same alias is used again.

- Aliases allow a general purpose application (such as the OpenEdge Data Dictionary) to expect a specific database name. The Dictionary only works on databases with logical name or alias “DICTDB”. The end user or the application can use CREATE ALIAS to provide the correct alias, in case it is inconvenient to connect the database using the correct logical name. Also, if there are several connected databases, the application can ask the user which one to select, then set the alias accordingly. The Data Dictionary does this when you choose Select Working Database.

- Suppose you connect to a database with logical name MYNAME and compile a procedure that accesses that database. Normally, the saved r-code file contains references to MYNAME.

  In a later session, when you want to use the precompiled program, you can connect to your database with the same logical name (MYNAME), or you can connect with a different logical name and set up an alias with the statement CREATE ALIAS “MYNAME” FOR DATABASE *logical name*.

- Usually, any alias that exists during the session when you compile a procedure has no effect on the resulting r-code file. When a procedure is compiled, the logical name of the database that is accessed within the procedure is put into the r-code file, not an existing alias. If a procedure accesses more than one database, all of the logical names of accessed databases are placed into the r-code file.

  However, any file reference that is qualified with an alias (as opposed to a logical name) generates a new instance of the file for the compilation. This new instance causes the r-code to have the alias reference and not the logical database name reference. Subsequent unqualified references to that same file within the same block, or nested blocks, will resolve to the new alias instance following the usual rules for qualifying. Unqualified references to different files in the same database do not get the alias name, but get the logical name. Anonymous references to a file, previously referenced using the alias qualifier, in a different, non-nested block get the logical name instead of the alias name.

It is simpler to just connect to a database with the desired logical name, leave all references unqualified, not create an alias, and then compile the application. However, sometimes you cannot precompile. In those cases, if you want to compile a procedure so that only the alias gets into the r-code file, then explicitly qualify all file references using the alias. You might want only the alias to get into the r-code file, so you can compile and distribute procedures that will run against any database whose logical name has been assigned the alias contained in the r-code file.

- Changes made to an alias do not take effect within the current procedure. In the following example, `alias1.p` fails to compile when it reaches the FOR EACH statement, because alias myalias has not been created during the compilation:

```
/* alias1.p */
/* NOTE: this code does NOT work */
CREATE ALIAS myalias FOR DATABASE sports.
FOR EACH myalias.customer:
  DISPLAY name.
END.
```

To solve this problem, split `r-alias1.p` into two procedures. For example:

r-dispnm.p

```
FOR EACH myalias.customer:
  DISPLAY name.
END.
```

r-alias2.p

```
CREATE ALIAS myalias FOR DATABASE sports.
RUN r-dispnm.p.
```

CREATE ALIAS affects only subsequent compilations; currently executing procedures are not affected.

- Be careful when using shared buffers with aliases. If you reference a shared buffer after changing the alias that initially was used in defining it, Progress returns a run-time error. See the following example procedures for details.

Once procedure r-main.p is run, it calls r-makebf.p, which calls r-disp6.p. The alias myalias is created in r-main.p, with reference to database sports. In r-makebf.p, the shared buffer mybuf is defined for myalias.customer. Then, in the next line, myalias is changed, so that it now refers to database sports2. When an attempt is made to reference shared buffer mybuf in procedure r-disp6.p, a run-time error occurs, with the message: "r-disp6.p Unable to find shared buffer for mybuf."

r-main.p

```
CREATE ALIAS myalias FOR DATABASE sports.
RUN r-makebf.p.
```

r-makebf.p

```
DEFINE NEW SHARED BUFFER mybuf FOR myalias.customer.
CREATE ALIAS myalias FOR DATABASE sports2.
RUN r-disp6.p
```

r-disp6.p

```
DEFINE SHARED BUFFER mybuf FOR myalias.customer.
FOR EACH mybuf:
  DISPLAY mybuf.
END.
```

See also ALIAS function, CONNECT statement, CONNECTED function, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, ERROR-STATUS system handle, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

CREATE automation object statement (Windows only)

Creates (instantiates) an ActiveX Automation object based on a specified Automation Server connection.

Syntax

```
CREATE expression1 COM-hdl-var
  [ CONNECT [ TO expression2 ] ]
  [ NO-ERROR ]
```

expression1

A character-string expression that evaluates to 1) a unique name of a valid Automation object stored in the system registry or 2) the null string ("").

COM-hdl-var

A COM-HANDLE variable that receives the COM handle to the instantiated Automation object.

[CONNECT **[** TO *expression2* **]]**

Specifies the connection option, together with *expression1*. The behavior of each connection option depends on the execution status of the Automation Server.

Table 18 illustrates this behavior.

Table 18: Automation object connection options *(1 of 3)*

Connection option	Server execution status	Connection behavior
1. Option omitted	Running	Creates a new instance of the Automation object identified by *expression1*. Launches a new instance of the Server for top-level Automation objects (like Excel.Application) start a new instance of the Server.
	Not running	Launches a new instance of the Server, then creates a new instance of the Automation object identified by *expression1*. Often, both the new Server and the new Automation object instance are invisibly created.
2. CONNECT	Running	Connects to an active (instantiated) Automation object identified by *expression1*. Works for top-level Automation objects only. For example, this works for Excel.Application but fails for Excel.Sheet and Excel.Chart, which are both lower-level Automation objects.
	Not running	Invalid. Always returns an error.

Table 18: Automation object connection options *(2 of 3)*

Connection option	Server execution status	Connection behavior
3. CONNECT TO *expression2*	Running	Creates or connects to an Automation object specified by *expression1* that is associated with the file specified by the pathname in *expression2*. If more than one instance of the Server is running, this option randomly selects one (generally, the first one started). If the specified file is already open within the selected Server, this option connects to the Automation object that is instantiated for that file. If the file is not already open in the selected Server, this option opens the file and instantiates the specified Automation object for it. If the specified file is already open in a different instance of the Server, this option fails with a "File in Use" error. This option also fails if the *expression2* does not specify a valid file.
	Not running	Creates a new instance of an Automation object specified by *expression1* that is associated with the file specified by the pathname in *expression2*. This option starts a new instance of the Server and instantiates the Automation object for the class that is initialized from the contents of the file. Often, the new Server, as well as the new Automation object, are invisibly created. This option fails if *expression2* does not specify a valid file.

Table 18: Automation object connection options *(3 of 3)*

Connection option	Server execution status	Connection behavior
4. CONNECT TO *expression2* WHERE *expression1* = ""	Running	Creates or connects to an Automation object that is associated with the file specified by the pathname in *expression2*. This option determines the identity of the Server (and hence the Automation object) from the file extension given in *expression2*. If more than one instance of the Server is running, this option randomly selects one (generally, the first one started). If the specified file is already open within the selected Server, this option connects to the Automation object that is instantiated for that file. If the file is not already open in the selected Server, this option opens the file and instantiates the specified Automation object for it. If the specified file is already open in a different instance of the Server, this option fails with a "File in Use" error. This option also fails if the *expression2* does not specify a valid file.
	Not running	Creates a new instance of an Automation object that is associated with the file specified by the pathname in *expression2*. This option determines the identity of the Server (and hence the Automation object) from the file extension given in *expression2*. This option starts a new instance of the Server and instantiates the Automation object for the class that is initialized from the contents of the file. Often, the new Server, as well as the new Automation object, are invisibly created. This option fails if *expression2* does not specify a valid file.

NO-ERROR

Suppresses error messages for the instantiation of an Automation object. You can then test for the ERROR condition to verify that the Automation object is instantiated.

Example

The following procedure demonstrates several Automation object instantiations using the four basic connection options. It tries all of the options with the Microsoft® Excel Automation Server. Note that not all Automation Servers support all options. For example in Office 95, there is no Automation object for PowerPoint presentations. Thus, the file connection option (Option 3 in Table 18) does not work.

r-crea.p *(1 of 3)*

```
/*
 * Demonstration of connecting to an Automation Object in Excel
 * using the different connection options.
 */

DEF BUTTON bExit
    LABEL "Exit" SIZE 16 BY 1.25 AUTO-GO.
DEF BUTTON bStart
    LABEL "Option 1 - Start Excel" SIZE 32 BY 1.25 .
DEF BUTTON bConnect
    LABEL "Option 2 - Connect to Active" SIZE 32 BY 1.25.
DEF BUTTON bConPerFile
    LABEL "Option 3 - Connect per File" SIZE 32 BY 1.25.
DEF BUTTON bConnectMon
    LABEL "Option 4 - Connect by Extension" SIZE 32 BY 1.25.
DEF VAR e AS CHAR VIEW-AS EDITOR SIZE 63 BY 1 LABEL "Result:" FONT 2.

DEFINE VARIABLE curDir AS CHARACTER.
FILE-INFO:FILE-NAME = ".".
curDir = FILE-INFO:FULL-PATHNAME.
DEFINE VAR wordAppl AS COM-HANDLE.


FORM e SKIP(0.5) bStart  SPACE bConnect  SPACE bConPerFile
                         SPACE bConnectMon
       SKIP(0.5) bExit WITH FRAME a VIEW-AS DIALOG-BOX THREE-D FONT 6.
FRAME a:TITLE = "Testing CREATE Automation Object Statement".

ENABLE ALL WITH FRAME a.ON CHOOSE OF bStart IN FRAME a
```

r-crea.p *(2 of 3)*

```
DO:

/*
 * Option 1:
 * Connect using CREATE expression1 Com-Handle-Var.
 */

  DEFINE VARIABLE excelAppl AS COM-HANDLE.
  CREATE "Excel.Application" excelAppl.
  excelAppl:Visible=true.
  excelAppl:Workbooks:Add.
  excelAppl:Range("A1"):Value = "testing CREATE".
  ASSIGN e:SCREEN-VALUE = String(excelAppl:Range("A1"):Value).
  release object excelAppl.
END.

ON CHOOSE OF bConnect IN FRAME a
DO:

/*
 * Option 2:
 * Connect using CREATE expression1 Com-Handle-Var CONNECT.
 */

  DEFINE VARIABLE excelAppl AS COM-HANDLE.
  CREATE "Excel.Application" excelAppl connect.
  excelAppl:Range("A2"):Value = "testing CONNECT".
  MESSAGE "Click me to continue!" VIEW-AS ALERT-BOX.
  ASSIGN e:SCREEN-VALUE = String(excelAppl:Range("A2"):Value).
  excelAppl:Workbooks:Item(1):SaveAs(curDir + "\zzz.xls").
  excelAppl:Quit().
  release object excelAppl.
END.
```

r-crea.p *(3 of 3)*

```
ON CHOOSE OF bStart IN FRAME a
DO:

/*
 * Option 1:
 * Connect using CREATE expression1 Com-Handle-Var.
 */

  DEFINE VARIABLE excelAppl AS COM-HANDLE.
  CREATE "Excel.Application" excelAppl.
  excelAppl:Visible=true.
  excelAppl:Workbooks:Add.
  excelAppl:Range("A1"):Value = "testing CREATE".
  ASSIGN e:SCREEN-VALUE = String(excelAppl:Range("A1"):Value).
  release object excelAppl.
END.

ON CHOOSE OF bConnect IN FRAME a
DO:

/*
 * Option 2:
 * Connect using CREATE expression1 Com-Handle-Var CONNECT.
 */

  DEFINE VARIABLE excelAppl AS COM-HANDLE.
  CREATE "Excel.Application" excelAppl connect.
  excelAppl:Range("A2"):Value = "testing CONNECT".
  MESSAGE "Click me to continue!" VIEW-AS ALERT-BOX.
  ASSIGN e:SCREEN-VALUE = String(excelAppl:Range("A2"):Value).
  excelAppl:Workbooks:Item(1):SaveAs(curDir + "\zzz.xls").
  excelAppl:Quit().
  release object excelAppl.
END.
```

Notes

- You must ensure that any third-party Automation objects you want to instantiate are installed and correctly listed in the registry. For information on what Automation objects you can instantiate, see the documentation for the third-party product. Generally, these are the same Automation objects instantiated by the Visual Basic CreateObject and GetObject functions. You might also be able to view these Automation objects using the OpenEdge COM Object Viewer tool. For more information, see *OpenEdge Development: Programming Interfaces*.
- The instantiation of an Automation object depends on the implementation of the Automation Server itself. Any Server registered for multiple use (REGCLS_MULTIPLE_USE flag) launches a single instance of the Server that handles multiple Automation object instantiation requests. Any Server registered single use (REGCLS_SINGLE_USE flag) launches a new instance of the Server for each instantiated Automation object.
- The four connection options in Table 18 compare to the following Visual Basic function calls:
  - **Option 1** — CreateObject (*class*) or GetObject ("", *class*)
  - **Option 2** — GetObject (, *class*)
  - **Option 3** — GetObject (*pathname*, *class*)
  - **Option 4** — GetObject (*pathname*)
- Once you create or connect to an Automation object, you can reference its properties and methods.

See also RELEASE OBJECT statement

CREATE BROWSE statement (Windows only; Graphical interfaces only)

Creates a dynamic browse, either read-only or updateable. Browse columns are added with the ADD-LIKE-COLUMN, ADD-COLUMNS-FROM, and ADD-CALC-COLUMN methods. The query is specified through the QUERY attribute.

The dynamic updateable browse can only be a NO-ASSIGN browse—you must make all data assignments to the database.

Note: Does not apply to SpeedScript programming.

Syntax

```
CREATE BROWSE widget-handle
  [ IN WIDGET-POOL widget-pool-name ]
  [ ASSIGN { attribute=expression } ... ]
  [ trigger-phrase ]
```

widget-handle

A variable of type WIDGET-HANDLE that Progress sets to the value of the new widget handle.

IN WIDGET-POOL *widget-pool-name*

Specifies the widget pool in which the object is created. If you do not specify a widget pool, the object is created in the current default widget pool. The browse will go away when its widget pool goes away or when you do a DELETE OBJECT on it.

ASSIGN { *attribute = expression* } ...

Assigns specified values to attributes of the object. The *attribute* parameter must be the name of a valid attribute for the object and *expression* must evaluate to a valid value for that attribute.

trigger-phrase

A trigger phrase associated with the object. For more information, see the Trigger phrase reference entry.

Example The following example creates a dynamic browse and adds columns to it:

r-dynbrws.p *(1 of 2)*

```
/* r-dynbrws */
DEFINE VARIABLE name-hdl AS WIDGET-HANDLE.
DEFINE VARIABLE num-hdl AS WIDGET-HANDLE.
DEFINE VARIABLE address-hdl AS WIDGET-HANDLE.
DEFINE VARIABLE calc-col-hdl AS WIDGET-HANDLE.
DEFINE VARIABLE browse-hdl AS WIDGET-HANDLE.
DEFINE VARIABLE buff-field-hdl AS WIDGET-HANDLE.
DEFINE VARIABLE brws-col-hdl AS WIDGET-HANDLE.
DEFINE BUTTON btn-delete LABEL "Delete".
DEFINE BUTTON btn-quit LABEL "&Quit"  AUTO-ENDKEY.
DEFINE VARIABLE j AS INTEGER.

DEFINE FRAME MyFrame SKIP(10)
             btn-delete btn-quit
       WITH SIZE 80 BY 22.

DEFINE QUERY q1 FOR customer SCROLLING.
OPEN QUERY q1 FOR EACH customer NO-LOCK.

CREATE BROWSE browse-hdl
    ASSIGN TITLE = "Dynamic Browse"
           FRAME = FRAME MyFrame:HANDLE
           QUERY = QUERY q1:HANDLE
           X = 2
           Y = 2
           WIDTH = 74
           DOWN = 10
           VISIBLE = YES
           SENSITIVE = TRUE
           READ-ONLY = NO.
```

r-dynbrws.p *(2 of 2)*

```
ON row-display OF browse-hdl DO:
  IF VALID-HANDLE(calc-col-hdl) THEN
    calc-col-hdl:SCREEN-VALUE =
                 STRING(customer.credit-limit - customer.balance).
END.

num-hdl = browse-hdl:ADD-LIKE-COLUMN("customer.cust-num").
name-hdl = browse-hdl:ADD-LIKE-COLUMN("customer.name").
address-hdl = browse-hdl:ADD-LIKE-COLUMN("customer.address").
calc-col-hdl = browse-hdl:ADD-CALC-COLUMN("INT","->,>>>,>>9.99","","Credit
Left").

/* Refresh needs to be done if ADD-CALC-COLUMN is done after the browse
* is displayed. In ROW-DISPLAY trigger, we can only set the calc field's
* screen-value if the handle is set. And the handle is set after the
* ADD-CALC-COLUMN method is done. */

browse-hdl:refresh().
browse-hdl:EXPANDABLE = YES.

ON row-leave OF browse-hdl DO:
    IF browse-hdl:CURRENT-ROW-MODIFIED THEN DO:
        REPEAT j = 1 TO browse-hdl:NUM-COLUMNS:
            brws-col-hdl = browse-hdl:GET-BROWSE-COLUMN(j).
            IF brws-col-hdl:MODIFIED THEN DO:
                buff-field-hdl = brws-col-hdl:BUFFER-FIELD.
            /* if buff-field-hdl is unknown, this is a calculated field
                  and cannot be updated */
                IF buff-field-hdl NE ? THEN
                    buff-field-hdl:BUFFER-VALUE =
                                   brws-col-hdl:SCREEN-VALUE.
            END.
        END.
    END.
END.

ON CHOOSE OF btn-delete DO:   /*  LABEL "DeleteDynBrowse". */
    DELETE WIDGET browse-hdl.
END.

ON CHOOSE OF btn-quit DO:
    QUIT.
END.

ENABLE ALL WITH FRAME MyFrame.
WAIT-FOR CLOSE OF CURRENT-WINDOW.
```

Notes

- If the browse's height is set using the DOWN attribute and a browse column is added, the browse's height may change to ensure that the number of DOWN is preserved. This may be due to the addition of the horizontal scrollbar or the growth of the column header.
- If the browse's height is set using the HEIGHT attribute or through direct manipulation, and a browse column is added, the DOWN attribute may change to ensure that the specified height is preserved. This may be due to the addition of the horizontal scrollbar or the growth of the column header.
- The DISPLAY . . . WITH BROWSE *browse-name* statement cannot be used with a dynamic browse. Instead, the user must set the SCREEN-VALUE attributes.
- A dynamic browse's validation expression is restricted. It may not contain a CAN-FIND function. To reference the field, the FRAME-VALUE function must be used. The CAN-FIND function will still work for a static browse column.
- If a buffer-field is associated with a dynamic browse column, you should set the buffer-field's VALIDATE-EXPRESSION attribute before the dynamic browse column is added to the browser (via ADD-LIKE-COLUMN( )). The validation expression is compiled at this time. If the VALIDATE-EXPRESSION attribute is changed later, it is ignored.
- You can use the ASSIGN option to assign a widget ID value to the WIDGET-ID attribute for this object. If you have enabled application-defined widget IDs in your OpenEdge GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then Progress uses this widget ID when it creates the widget at runtime, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then Progress ignores this option setting at runtime.

  For more information about the WIDGET-ID attribute, see its reference entry in the "Attributes and Methods Reference" section on page 1497. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

See also

ADD-CALC-COLUMN( ) method, ADD-COLUMNS-FROM( ) method, ADD-LIKE-COLUMN( ) method, CREATE QUERY statement, CREATE widget statement, DEFINE BROWSE statement, DEFINE QUERY statement, GET-BROWSE-COLUMN( ) method, QUERY attribute

CREATE BUFFER statement

Creates a dynamic buffer.

Syntax

```
CREATE BUFFER handle FOR TABLE table-exp | table-handle-exp
  [ BUFFER-NAME buffer-expression ]
  [ IN WIDGET-POOL widget-pool-name ]
```

handle

A variable of type HANDLE that represents the handle of the buffer object.

FOR TABLE *table-exp* | *table-handle-exp*

A character expression that evaluates to a unique database table name or temp-table name or to the handle of a database table or a temp-table.

Note: If the table name is ambiguous, you must qualify it with a database name.

BUFFER-NAME *buffer-expression*

An expression of type CHARACTER that evaluates, at run time, to the name of the dynamic buffer you are creating. This option lets a dynamic query have multiple buffers for the same table.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the dynamic buffer.

Note: Widget pool names are not case-sensitive.

Example The following example runs the query “for each customer” dynamically against the Sports database using a purely dynamic buffer with no compile time references at all:

r-crtbuf.p

```
/* r-crtbuf.p */
/* requires a connection to the Sports database */

DEFINE VARIABLE i AS INTEGER.
DEFINE VARIABLE qh AS WIDGET-HANDLE.
DEFINE VARIABLE bh AS WIDGET-HANDLE.
DEFINE VARIABLE fh AS WIDGET-HANDLE EXTENT 10.

CREATE BUFFER bh FOR TABLE "customer".
CREATE QUERY qh.

qh:SET-BUFFERS(bh).
qh:QUERY-PREPARE("for each customer").
qh:QUERY-OPEN.
qh:GET-FIRST.

DISPLAY bh:NAME.

REPEAT i = 1 TO 10.
  fh[i] = bh:BUFFER-FIELD(i).
  DISPLAY fh[i]:NAME STRING(fh[i]:BUFFER-VALUE).
END.

DELETE WIDGET bh.
```

Note For more information on dynamic buffers, see *OpenEdge Development: Progress 4GL Handbook*.

See also CREATE QUERY statement, DEFINE BUFFER statement

CREATE CALL statement

Creates a CALL object, then stores a handle to it in the handle variable specified.

The CALL object, its attributes, and its methods, are used by applications to invoke logic dynamically.

Syntax

```
CREATE CALL handle [ IN widget-pool ] [ NO-ERROR ]
```

handle

A HANDLE expression that indicates the name of a HANDLE variable into which a handle to the new CALL object is stored.

IN *widget-pool*

A CHARACTER expression that indicates the name of the widget pool to contain the new CALL object.

NO-ERROR

Suppresses reporting of errors that occur while CREATE CALL executes. Afterwards, you can get information on possible errors by checking the ERROR-STATUS system handle.

Notes

- Unlike most 4GL objects, the CALL object, by default, is assigned not to the closest unnamed widget pool, but rather to the SESSION widget pool.
- A CALL object is deleted automatically when its widget pool is deleted. To delete it earlier than its widget pool, use the DELETE OBJECT statement, specifying the handle to the CALL object, as in the following fragment:

```
DELETE OBJECT hMyCallObj.
```

See also

CALL object handle, DELETE OBJECT statement

CREATE CLIENT-PRINCIPAL statement

Creates an instance of a Client-principal object dynamically at run time. Each Client-principal object instance contains information specific to one user login session. This login session may be used as an application or database user identity from an authentication domain registered in the application's trusted authentication domain registry.

Note: You use a Client-principal object with the SET-CLIENT( ) method or SET-DB-CLIENT function to set the user identity for an OpenEdge session or OpenEdge database. You can have only one active Client-principal object set as the current user at any one point in time for a session or database connection.

Syntax

```
CREATE CLIENT-PRINCIPAL client-principal-handle
```

client-principal-handle

A variable of type HANDLE that represents the handle of the Client-principal object.

Note

To use the Client-principal object, you must first register the authentication domain that created the object in the application's trusted authentication domain registry for the session or database connection. To register an authentication domain, you can do either of the following:

- Use the REGISTER-DOMAIN( ) and LOCK-REGISTRATION( ) methods, which register an authentication domain and then restrict the registration of additional authentication domains, respectively.
- Use the LOAD-DOMAINS( ) method, which loads registered authentication domains from an OpenEdge database and then automatically restricts the registration of additional authentication domains.

See also

Client-principal object handle, LOAD-DOMAINS( ) method, LOCK-REGISTRATION( ) method, REGISTER-DOMAIN( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function

CREATE DATABASE statement

Creates a new OpenEdge database.

Syntax

```
CREATE DATABASE new-database [ FROM old-database [ NEW-INSTANCE ] ]
  [ REPLACE ] [ NO-ERROR ]
```

new-database

A CHARACTER expression that returns the full or relative pathname of the database you want to create. If the database already exists, a new database is not created unless you specify REPLACE.

FROM *old-database*

A CHARACTER expression that returns the name of the database whose schema and data you want to copy to the new database. The value of *old-database* can be a full or relative pathname or one of the special strings "EMPTY", "DEMO", or "SPORTS". If you omit this option, Progress creates an empty database.

NEW-INSTANCE

If specified, Progress assigns the new database a new globally unique identifier (GUID) value as the database identifier. If not specified, Progress assigns the new database the same GUID database identifier as the old database.

When you create a new database by copying an existing 10.1A database provided by Progress (such as the empty database, demo database, or Sports database), Progress always assigns the new database a new GUID database identifier.

Note: Use this option only when creating a new 10.1A database by copying an existing 10.1A database.

REPLACE

If specified and a database already exists with the name specified by *new-database*, the existing database is deleted and replaced with the new database. If not specified and a database already exists with the name specified by *new-database*, an error occurs.

NO-ERROR

If specified and the CREATE DATABASE statement fails, the error condition is not raised.

Example

This procedure prompts for the name of a database to connect. If the database does not exist, the procedure creates it.

r-credb.p

```
DEFINE VARIABLE dbname AS CHARACTER LABEL "Database" FORMAT "x(65)".

/* Prompt the user for the name of a demo database to connect. */
SET dbname HELP "Enter the name of your database."
    WITH FRAME dbname-frame SIDE-LABELS.

/* If the entered name does not have the .db suffix, add it.
   This is necessary for the search function to work correctly. */
IF LENGTH(dbname) < 3
THEN dbname = dbname + ".db".
ELSE IF SUBSTR(dbname, LENGTH(dbname) - 2) = ".db"
     THEN dbname = dbname + ".db".

/* If the database does not exist, create it from SPORTS. */
IF SEARCH(dbname) = ?
THEN DO:
  MESSAGE "Database does not exist. Do you want to create it?"
     VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO TITLE "Connect Database"
       UPDATE create-it AS LOGICAL.

  IF create-it
  THEN DO:
    CREATE DATABASE dbname FROM "SPORTS".
    MESSAGE "New database created:" dbname.
  END.
  ELSE UNDO, RETRY.
END.

/* Connect the database. */
CONNECT VALUE(dbname) -1.
```

Notes

- If you omit the FROM option, Progress uses the empty database.
- If you use the NO-ERROR option, you can use the ERROR-STATUS system handle to obtain information on errors that occurred in processing the CREATE DATABASE statement.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DELETE ALIAS statement, DISCONNECT statement, ERROR-STATUS system handle, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

CREATE DATASET statement

Creates a ProDataSet object dynamically at run time. The ProDataSet object that is created is empty.

Note: You can use the ADD-BUFFER( ) method and SET-BUFFERS( ) method to add buffers to a dynamic ProDataSet object.

Syntax

```
CREATE DATASET dataset-handle [ IN WIDGET-POOL widget-pool-name ]
```

dataset-handle

A variable of type HANDLE that represents the handle of the dynamic ProDataSet object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool in which the dynamic ProDataSet object is created.

Note: Widget pool names are not case-sensitive.

Notes

- If you do not specify a widget pool name, the object is created in the Session unnamed widget pool (not in the closest unnamed widget pool). The object goes away when its widget pool goes away or when you delete it using the DELETE OBJECT statement.
- If the ProDataSet object serves as an OUTPUT parameter and you specify a widget pool, the widget pool must outlive the called procedure.

Example

Following is an example of how to create a dynamic ProDataSet object for Orders and their Orderlines:

```
DEFINE VARIABLE hDset AS HANDLE NO-UNDO.
DEFINE VARIABLE hRel  AS HANDLE NO-UNDO.

CREATE DATASET hDset.

hDset:SET-BUFFERS(BUFFER ttOrder:HANDLE, BUFFER ttOrderLine:HANDLE).

hRel = hDset:ADD-RELATION(BUFFER ttOrder:HANDLE,
     BUFFER ttOrderLine:HANDLE, "ttOrder.OrderNum,ttOrderLine.OrderNum").
```

See also

ProDataSet object handle, DEFINE DATASET statement

CREATE DATA-SOURCE statement

Creates a data-source object dynamically at run time.

Syntax

```
CREATE DATA-SOURCE data-source-handle [ IN WIDGET-POOL widget-pool-name ]
```

data-source-handle

A variable of type HANDLE that represents the handle of the dynamic data-source object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool in which the dynamic data-source object is created.

Note: Widget pool names are not case-sensitive.

Note

If you do not specify a widget pool name, the dynamic data-source object is created in the closest unnamed widget-pool, by default. The object goes away when its widget pool goes away or when you delete it using the DELETE OBJECT statement.

See also

Data-source object handle, DEFINE DATA-SOURCE statement

CREATE QUERY statement

Creates a dynamic query.

Syntax

```
CREATE QUERY handle
 [ IN WIDGET-POOL widget-pool-name ]
```

handle

A variable of type HANDLE that represents the handle of the query object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the dynamic query.

Note: Widget pool names are not case-sensitive.

Example

The following example creates a dynamic query with a static buffer and a dynamic predicate (WHERE clause) which is resolved at run time:

r-crtqry.p

```
/* r-crtqry.p */

DEFINE VARIABLE qh AS WIDGET-HANDLE.
DEFINE VARIABLE numvar AS INTEGER INITIAL 10.

CREATE QUERY qh.

qh:SET-BUFFERS(BUFFER customer:HANDLE).
qh:QUERY-PREPARE("FOR EACH customer WHERE cust-num < " + string(numvar)).
qh:QUERY-OPEN.

REPEAT WITH FRAME y:
  qh:GET-NEXT().
  IF qh:QUERY-OFF-END THEN LEAVE.
  DISPLAY cust-num
          name      FORMAT "x(30)"
          city      FORMAT "X(20)"
END.

qh:QUERY-CLOSE()
DELETE OBJECT qh.
```

Notes

- CREATE-QUERY must be followed by the QUERY-PREPARE( ) and QUERY-OPEN() methods before the query can be run.
- For more information on dynamic queries, see *OpenEdge Development: Progress 4GL Handbook*.

See also

CREATE BUFFER statement, DEFINE QUERY statement, QUERY-OPEN( ) method, QUERY-PREPARE( ) method

CREATE SAX-READER statement

Creates an instance of a SAX-reader object and assigns its handle to the handle variable you specify. The SAX-reader object can be used to control the parsing of XML source.

Syntax

```
CREATE SAX-READER handle
  [ IN WIDGET-POOL pool-name ] [ NO-ERROR ]
```

handle

A variable of type HANDLE into which CREATE SAX-READER stores the new handle.

IN WIDGET-POOL *pool-name*

A CHARACTER expression indicating the widget pool in which the object is created. If you do not specify a widget pool, the object is created in the current default widget pool.

NO-ERROR

Causes any errors that occur during the creation of the SAX-reader handle to be suppressed. After the CREATE SAX-READER statement completes, check the ERROR-STATUS handle for information concerning any errors that might have occurred.

See also

SAX-reader object handle

CREATE SAX-WRITER statement

Creates an instance of a SAX-writer object and assigns its handle to the handle variable specified. Use this object to write an XML document using the SAX interface.

Syntax

```
CREATE SAX-WRITER handle [ IN WIDGET-POOL pool-name ] [ NO-ERROR ]
```

handle

Variable of type HANDLE which stores the handle of the new SAX-writer object.

IN WIDGET-POOL *pool-name*

Specifies the widget pool where Progress creates the new object. If you do not specify a widget pool, Progress creates the object in the current default widget pool.

NO-ERROR

Specifies that Progress should suppress errors occurring during the creation of the SAX-writer handle. After the CREATE SAX-WRITER statement completes, you can check the ERROR-STATUS system handle for information about errors that might have occurred.

See also

SAX-writer object handle

CREATE SERVER statement

Creates an instance of a server object and assigns its handle to the handle variable you specify.

Syntax

```
CREATE SERVER handle
  [ ASSIGN { attribute = expression } ... ]
```

handle

A variable of type HANDLE into which CREATE SERVER stores the new server handle.

ASSIGN { *attribute = expression* } ...

Assigns specified values to attributes of the handle. The *attribute* parameter must be the name of a valid attribute for a server handle, and the *expression* parameter must evaluate to a valid value for the attribute.

Note

You can use a server handle as a connection point to an AppServer™. For more information on server handles, see the Server object handle entry. For more information on AppServers, see *OpenEdge Application Server: Developing AppServer Applications*.

See also

DELETE OBJECT statement, RUN statement, Server object handle

CREATE SERVER-SOCKET statement

Creates an instance of a server socket object and assigns it to the handle variable specified. It is through this object that a socket-based server application can listen for connections on a TCP/IP port.

Note: Does not apply to SpeedScript programming.

Syntax

```
CREATE SERVER-SOCKET handle [ NO-ERROR ]
```

handle

Variable of type HANDLE into which the CREATE SERVER-SOCKET statement stores the new server socket handle.

NO-ERROR

Specifies that any errors that occur during the creation of the server socket handle are suppressed. After the CREATE SERVER-SOCKET statement completes, the ERROR-STATUS system handle can be checked for information about any errors that might have occurred.

Notes

- An application can only create one server socket object. This statement will raise ERROR if an application tries to create multiple objects.
- A server socket object cannot be used with an AppServer or WebSpeed agent.

See also

CREATE SOCKET statement, DELETE OBJECT statement, Server socket object handle, Socket object handle

CREATE SOAP-HEADER statement

Creates an instance of a SOAP-header object dynamically at run time, and assigns its handle to the specified handle variable.

Syntax

```
CREATE SOAP-HEADER handle [ IN WIDGET-POOL widget-pool-name ]
```

handle

A variable of type HANDLE that represents the handle of the SOAP-header object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates to the name of the widget pool in which the dynamic SOAP-header object is created.

Note: Widget pool names are not case-sensitive.

Notes

- Use the SOAP-header object to pass an input parameter to a response callback procedure and an output parameter to a request callback procedure. The SOAP HEADER object passed to the response callback is implicitly created by OpenEdge. In order to pass a SOAP HEADER object back from the request callback, the application needs to explicitly create it or use an object that it has previously saved.
- The SOAP-header object is either implicitly created by Progress or explicitly created by the application using the CREATE SOAP-HEADER statement. In either case, the application is responsible for deleting the object.

  You can delete a SOAP-header object and its underlying XML in one of two ways. You can:

  – Use the DELETE OBJECT statement to delete the SOAP-header object directly.

  – Set the `lDeleteOnDone` parameter in the request header callback procedure to TRUE, which directs OpenEdge to delete the SOAP header object after OpenEdge copies the object's contents to the outbound SOAP message.

  For more information about deleting SOAP-header objects, see *OpenEdge Development: Web Services*.

See also

SOAP-header object handle

CREATE SOAP-HEADER-ENTRYREF statement

Creates an instance of a SOAP-header-entryref object dynamically at runtime, and assigns its handle to the specified handle variable.

Syntax

```
CREATE SOAP-HEADER-ENTRYREF hshEntry [ IN WIDGET-POOL widget-pool-name ]
```

hshEntry

A variable of type HANDLE that represents the handle of the SOAP-header-entryref object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates to the name of the widget pool in which the dynamic SOAP-header-entryref object is created. A SOAP-header-entryref object is created in the closest unnamed widget-pool, by default.

Note: Widget pool names are not case-sensitive.

Note

The SOAP-header-entryref object is explicitly created by the application using the CREATE SOAP-HEADER-ENTRYREF statement. The application is responsible for deleting this object. Use the DELETE OBJECT statement to delete a SOAP-header-entryref object without deleting its underlying XML. Use the DELETE-HEADER-ENTRY() method to delete the XML underlying the SOAP-header-entryref object without deleting the object.

See also

SOAP-header-entryref object handle

CREATE SOCKET statement

Creates a socket object and assigns it to the handle variable specified. It is through this object that the application can connect to a TCP/IP port and read and write on the socket bound to the port.

Syntax

```
CREATE SOCKET handle [ NO-ERROR ]
```

handle

Variable of type HANDLE into which CREATE SOCKET stores the new socket handle.

NO-ERROR

Specifies that any errors that occur during the creation of the socket handle are suppressed. After the CREATE SOCKET statement completes, the ERROR-STATUS system handle can be checked for information about any errors that might have occurred.

See also CREATE SERVER-SOCKET statement, DELETE OBJECT statement, Server socket object handle, Socket object handle

CREATE TEMP-TABLE statement

Creates a temp-table dynamically at run time. The temp-table that is created is empty and must be defined using ADD/CREATE methods.

Syntax

```
CREATE TEMP-TABLE widget-handle
 [ IN WIDGET-POOL widget-pool-name ]
```

widget-handle

A variable of type WIDGET-HANDLE that represents the handle of the temp-table object.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the dynamic temp-table.

Note: Widget pool names are not case-sensitive.

Example

The following example creates a temp-table like the order table and populates it from the order table. In addition, the corresponding sales-rep name is added from the salesrep table.

r-cretmpt.p *(1 of 2)*

```
DEFINE VARIABLE tth AS HANDLE.
DEFINE VARIABLE bh AS HANDLE.
DEFINE VARIABLE qh AS HANDLE.
DEFINE VARIABLE buf-ord-hndl AS HANDLE.
DEFINE VARIABLE buf-rep-hndl AS HANDLE.
DEFINE VARIABLE fld1 AS HANDLE.
DEFINE VARIABLE fld2 AS HANDLE.

/* get database table handles */
buf-ord-hndl = BUFFER order:HANDLE.
buf-rep-hndl = BUFFER salesrep:HANDLE.

/* create an empty undefined temp-table */
CREATE TEMP-TABLE tth.
/* give it order table's fields & indexes */
tth:CREATE-LIKE(buf-ord-hndl).
/* add field like Salesrep.Rep-Name */
tth:ADD-LIKE-FIELD("RepName","Salesrep.Rep-Name").
/* no more fields will be added */
tth:TEMP-TABLE-PREPARE("ordx").

/* get the buffer handle for the temp-table */
bh = tth:DEFAULT-BUFFER-HANDLE.

/* populate the temp-table from order */
FOR EACH order:
    bh:BUFFER-CREATE.
    bh:BUFFER-COPY(buf-ord-hndl).
/* add the corresponding salesrep name */
    FIND salesrep WHERE salesrep.sales-rep = order.sales-rep NO-ERROR.
    IF AVAILABLE salesrep THEN
        bh:BUFFER-COPY(buf-rep-hndl,?,"RepName,rep-name").
END.

/* run a query to access the temp-table */
CREATE QUERY qh.
qh:SET-BUFFERS(bh).
qh:QUERY-PREPARE("for each ordx where order-num < 50 BY RepName").
qh:QUERY-OPEN().
```

r-cretmpt.p *(2 of 2)*

```
fld1 = bh:BUFFER-FIELD("order-num").
fld2 = bh:BUFFER-FIELD("RepName").

/* display the order-number and the salesrep name */
REPEAT:
    qh:GET-NEXT().
    IF qh:QUERY-OFF-END THEN LEAVE.
    DISPLAY fld1:BUFFER-VALUE() FORMAT "X(10)".
    DISPLAY fld2:BUFFER-VALUE() FORMAT "X(20)".
END.

qh:QUERY-CLOSE().
bh:BUFFER-RELEASE().
DELETE OBJECT tth.
DELETE OBJECT qh.
```

Notes

- Once the temp-table fields and indexes are defined using the ADD/CREATE methods, the definition must be terminated by using the TEMP-TABLE-PREPARE method before the temp-table can be used.
- Once the temp-table is prepared, it can be manipulated by using its buffer object handle which is retrieved using the DEFAULT-BUFFER-HANDLE attribute. All the BUFFER methods are available to the dynamic temp-table.
- The dynamic temp-table object is scoped like the buffer object. It is created in a widget pool and ends when the widget pool ends or when it is deleted with the DELETE OBJECT statement. You may not delete the default buffer object belonging to a dynamic temp-table.
- Errors for dynamic temp-tables do not automatically raise the ERROR condition since they occur inside a widget expression. All the methods that can have errors return FALSE if an error occurs, so they must be tested. If NO-ERROR is in effect in the statement containing the widget reference, no messages display, but they can be retrieved from the ERROR-STATUS system handle.

See also DEFINE TEMP-TABLE statement, TEMP-TABLE-PREPARE( ) method

CREATE widget statement

Creates a dynamic object, such as a widget object.

Note: Does not apply to SpeedScript programming.

Syntax

```
CREATE { BUTTON | COMBO-BOX
       | CONTROL-FRAME | DIALOG-BOX
       | EDITOR | FILL-IN
       | FRAME | IMAGE
       | MENU | MENU-ITEM
       | RADIO-SET | RECTANGLE
       | SELECTION-LIST | SLIDER
       | SUB-MENU | TEXT
       | TOGGLE-BOX | WINDOW
       | VALUE ( string-expression )
       } widget-handle [ IN WIDGET-POOL pool-name ]
  [ ASSIGN { attribute = expression } ... ]
  [ trigger-phrase ]
```

`VALUE ( string-expression )`

An expression of type CHARACTER that evaluates to the type of object you want to create (for example, BUTTON) with any combination of uppercase and lowercase characters.

widget-handle

A variable of type WIDGET-HANDLE that Progress sets to the value of the new widget handle.

`IN WIDGET-POOL pool-name`

Specifies the widget pool in which the object is created. If you do not specify a widget pool, the object is created in the current default widget pool.

`ASSIGN { attribute = expression } ...`

Assigns specified values to attributes of the object. The *attribute* parameter must be the name of a valid attribute for the object and *expression* must evaluate to a valid value for that attribute.

trigger-phrase

A trigger phrase associated with the object. For more information, see the Trigger phrase reference entry.

Example

This procedure creates a dynamic button that displays a list of customer names:

r-dynbut.p

```
DEFINE VARIABLE  but1  AS WIDGET-HANDLE.
DISPLAY "Dynamic Button Example" SKIP(3) WITH FRAME x SIDE-LABELS.

OPEN QUERY all-custs FOR EACH Customer.
GET FIRST all-custs.
DISPLAY Customer.Name WITH FRAME x.

CREATE BUTTON but1
   ASSIGN ROW = 3
   COLUMN = 5
   LABEL = "Next Customer"
   FRAME = FRAME x:HANDLE
   SENSITIVE = TRUE
   VISIBLE = TRUE
    TRIGGERS:
       ON CHOOSE
           DO:
              GET NEXT all-custs.
              DISPLAY Customer.Name WITH FRAME x.
           END.
    END TRIGGERS.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- Attribute assignments you specify in the CREATE widget statement are processed in the order they appear. In some cases you must supply the attributes in proper order. For example, you cannot set the SENSITIVE or VISIBLE attributes for a field-level widget until you have set its FRAME attribute.
- If you are setting the FORMAT attribute and specifying an initial SCREEN-VALUE for the widget, assign the FORMAT before the SCREEN-VALUE. Otherwise, the default format is applied to the SCREEN-VALUE which might cause truncation or other formatting errors.
- You can use the ASSIGN option to assign a widget ID value to the WIDGET-ID attribute for this object. If you have enabled application-defined widget IDs in your OpenEdge GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then Progress uses this widget ID when it creates the widget at runtime, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then Progress ignores this option setting at runtime.

  For more information about the WIDGET-ID attribute, see its reference entry in the "Attributes and Methods Reference" section on page 1497. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- If you create a frame to use as a DDE frame, you must realize the frame (set its VISIBLE attribute to TRUE) before using it as a conversation end-point. If you want the DDE frame to remain invisible during its use in a DDE conversation, set its HIDDEN attribute to TRUE after realizing the frame. For information on DDE frames, see *OpenEdge Development: Programming Interfaces*.

See also

CREATE QUERY statement, CREATE WIDGET-POOL statement, DEFINE FRAME statement, DELETE WIDGET statement, DELETE WIDGET-POOL statement, Trigger phrase

CREATE WIDGET-POOL statement

Creates a named or unnamed widget pool.

Note: Does not apply to SpeedScript programming.

Syntax

```
CREATE WIDGET-POOL
  [ pool-name [ PERSISTENT ] ]
  [ NO-ERROR ]
```

pool-name

A character-string expression specifying the name of the pool you are creating. If you omit this option, an unnamed pool is created. Widget pool names are not case sensitive.

PERSISTENT

Specifies that the pool is persistent. This means that the pool and any widgets in it remain allocated after the current procedure terminates. If you do not specify this option, the pool and its contents are automatically deleted when procedure execution ends.

NO-ERROR

Suppresses error messages if the specified widget pool already exists. You can then test for the ERROR condition to verify that the widget pool does, in fact, exist.

Example

The following example lets you create a series of dynamic buttons. All the buttons are created within a named widget pool. Because the widget pool is created within a trigger, it is defined as persistent so that it remains allocated after the trigger ends. You can at any time choose to delete the entire widget pool and start over.

r-widpl.p *(1 of 2)*

```
DEFINE VARIABLE wh AS WIDGET-HANDLE.

DEFINE BUTTON b_create LABEL "Create Button".
DEFINE BUTTON b_del    LABEL "Delete Buttons".
DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE
    DO:
      IF VALID-HANDLE(wh) THEN
        DELETE WIDGET-POOL "new-buttons".
      QUIT.
    END.
  END.

DEFINE FRAME butt-frame
  b_create b_del b_quit
  WITH ROW SCREEN-LINES - 2.

DEFINE FRAME new-buttons
  WITH SIZE 76 BY 11 CENTERED ROW 2 TITLE "New Buttons".

ON CHOOSE OF b_create IN FRAME butt-frame
DO:
  STATUS INPUT "Press RETURN to select a new button".
  IF wh = ? OR NOT VALID-HANDLE(wh) THEN
    CREATE WIDGET-POOL "new-buttons" PERSISTENT.
    CREATE BUTTON wh IN WIDGET-POOL "new-buttons"
    ASSIGN FRAME = FRAME new-buttons:HANDLE
      ROW = RANDOM(2, 9)
      COLUMN = RANDOM(2, 58)
      LABEL = "BUTTON " + STRING(etime)
      SENSITIVE = TRUE
      VISIBLE = TRUE
      TRIGGERS:
        ON CHOOSE PERSISTENT RUN dispmsg.
      END.
END.
```

r-widpl.p *(2 of 2)*

```
ON CHOOSE OF b_del IN FRAME butt-frame
DO:
  IF VALID-HANDLE(wh) THEN
    DELETE WIDGET-POOL "new-buttons".
  STATUS INPUT.
END.

ENABLE b_create b_del b_quit WITH FRAME butt-frame.

DO ON ENDKEY UNDO, LEAVE:
   WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.
END.

IF VALID-HANDLE(wh)
THEN DELETE WIDGET-POOL "new-buttons".

PROCEDURE dispmsg:
  MESSAGE "You chose button " SELF:LABEL.
END.
```

Notes

- Progress automatically creates a persistent unnamed widget pool at the start of each session. Most applications use only the session widget pool.
- Unnamed widget pools cannot be persistent, except the session widget pool, which is created by Progress.
- Persistent widget pools remain allocated until they are explicitly deleted (with the DELETE WIDGET-POOL statement) or until the end of the OpenEdge session that created them.
- All named widget pools are globally scoped. While a named widget pool is allocated, any procedure within the same process can access that widget pool. The name of a widget pool must be unique among all widget pools for the process. If you try to create a widget pool with the same name as an existing pool, Progress raises the ERROR condition.
- If a recursive procedure creates an unnamed widget pool, each iteration of that procedure creates a separate pool. If a recursive routine creates a named widget pool, you must ensure that only one iteration creates the pool (where all iterations can share it) or use a different name in each iteration (where each creates and uses its own pool).

- When you create an unnamed widget pool, it automatically becomes the default widget pool. This means that each subsequent dynamically created widget is placed in that pool unless you specifically assign it to another pool. The unnamed pool you create remains the default widget pool until it is deleted or you create another unnamed widget pool.
- You might want to create a new, unnamed widget pool just before invoking a new procedure and then delete that pool when the procedure returns. This ensures that any dynamic widgets created by that procedure in the default pool are deleted immediately. For example:

```
CREATE WIDGET-POOL.
RUN xyz.p.
DELETE WIDGET-POOL.
```

Similarly, you might want to store all dynamic widgets for a subsystem within a specific named pool. For example:

```
CREATE WIDGET-POOL "oe-pool".
RUN ord-ent.p
DELETE WIDGET-POOL "oe-pool".
```

In this example, the procedure ord-ent.p must reference the oe-pool for each dynamic widget it creates.

See also CREATE widget statement, DELETE WIDGET-POOL statement

CREATE X-DOCUMENT statement

Creates a handle for an XML document object. To use the XML document, you must add new nodes using the CREATE-NODE( ) method, the CREATE-NODE-NAMESPACE( ) method, or populate the document from an existing file using the LOAD( ) method.

Note: To ensure consistency across all nodes in an XML document, use either the CREATE-NODE-NAMESPACE( ) method or the CREATE-NODE( ) method to build an XML document; do not use both methods within a single document.

Syntax

```
CREATE X-DOCUMENT handle [ IN WIDGET-POOL widget-pool-name ]
```

handle

A variable of type HANDLE into which CREATE X-DOCUMENT stores the new handle.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the XML document object.

Note: Widget pool names are not case-sensitive.

Example

The following code fragment depicts creating an XML document object:

```
DEFINE VARIABLE hXdoc AS HANDLE.
. . .
CREATE X-DOCUMENT hXdoc.
. . .
```

See also

CREATE X-NODEREF statement, DELETE OBJECT statement, X-document object handle, X-noderef object handle

CREATE X-NODEREF statement

Creates a handle which can be used as a parameter or return-value for methods which will associate the handle with an XML node. This object is not a node in its own right, but merely a way to provide access to the underlying XML node.

Syntax

```
CREATE X-NODEREF handle [ IN WIDGET-POOL widget-pool-name ]
```

handle

A valid X-noderef object handle to use for the new XML node.

IN WIDGET-POOL *widget-pool-name*

An expression of type CHARACTER that evaluates, at run time, to the name of the widget pool that contains the XML node.

Note: Widget pool names are not case-sensitive.

Example

The following code fragment depicts creating an XML document node reference and using it to create a node:

```
DEFINE VARIABLE hXdoc AS HANDLE.
. . .
CREATE X-DOCUMENT hXdoc.
CREATE X-NODEREF hXnode.
hXdoc:CREATE-NODE(hXnode,"City","ELEMENT").
. . .
```

See also

CREATE X-DOCUMENT statement, DELETE OBJECT statement, X-document object handle, X-noderef object handle

CURRENT-CHANGED function

Returns TRUE if the copy of the record in the buffer after executing a FIND CURRENT or GET CURRENT differs from the copy of the record in the buffer before executing the FIND CURRENT or GET CURRENT. That is, if the current application changes the record, but no other user changes the record during its scope in the current application, CURRENT-CHANGED returns FALSE.

Syntax

```
CURRENT-CHANGED record
```

record

The name of a table or buffer.

Example

The following example finds the first customer record with NO-LOCK and makes it available to the user to review and change. While the user reviews the record, other users can change it. After the user makes a change of their own and enters GO in the frame, the first FIND CURRENT statement refetches the current customer record with an EXCLUSIVE-LOCK (preventing other users from reading or updating it). Then, the CURRENT-CHANGED function compares the contents of the customer record with the copy of the customer record that was in the buffer before the FIND CURRENT statement. If it differs, the CURRENT-CHANGED function returns a TRUE value, prints a message, and displays the contents of the customer record contained in the buffer. The RETURN NO-APPLY option prevents the program from ending and gives the user another chance to change the customer record.

The CURRENT-CHANGED function returns a FALSE value if the copy of the customer record that is in the buffer was not modified. After verifying that the copy of the record has not changed, the ASSIGN statement updates the customer record and a second FIND CURRENT statement down grades the record to NO-LOCK. Thus, while the user has ample time to review and change the record, the actual transaction time is kept to a minimum to allow other users access.

r-currch.p

```
FORM customer.name customer.balance WITH FRAME upd.

ON GO OF FRAME upd DO:
    DO TRANSACTION:
        FIND CURRENT customer EXCLUSIVE-LOCK.
        IF CURRENT-CHANGED customer THEN DO:
            MESSAGE "This record has been changed by another user"
                    SKIP
                    "Please re-enter your changes."
                    VIEW-AS ALERT-BOX.
            DISPLAY customer.name customer.balance with frame upd.
            RETURN NO-APPLY.
        END.
        ASSIGN customer.name customer.balance.
    END.
    FIND CURRENT customer NO-LOCK.
END.

FIND FIRST customer NO-LOCK.
DISPLAY customer.name customer.balance WITH FRAME upd.
DO ON ENDKEY UNDO, LEAVE:
    ENABLE customer.name customer.balance WITH FRAME upd.
    WAIT-FOR "GO" OF FRAME upd.
END.
```

Notes

- The CURRENT-CHANGED function is valid only when called after a FIND CURRENT or GET CURRENT statement.
- If a client application modifies the buffer, Progress compares the newly read record with the buffer contents from that application, rather than with the record read from the server. The CURRENT-CHANGED function continues to return a value based on the contents of the buffer until the next FIND CURRENT or GET CURRENT operates on that buffer or until the buffer goes out of scope or is released.

See also FIND statement, GET statement, LOCKED function

CURRENT-LANGUAGE function

Returns the current value of the CURRENT-LANGUAGE variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
CURRENT-LANGUAGE
```

Example

The following example displays a message indicating the setting of your CURRENT-LANGUAGE:

r-curlng.p

```
DEFINE VARIABLE cur-lang AS CHARACTER.

cur-lang = CURRENT-LANGUAGE.

IF cur-lang = "?"
THEN MESSAGE "Your current language is not set.".
ELSE MESSAGE "Your current language is" cur-lang.
```

Notes

- An r-code file may contain several text segments each associated with a different language. The setting of the CURRENT-LANGUAGE variable determines from which r-code text segment Progress reads character-string constants.
- If the value of CURRENT-LANGUAGE is a quoted question mark ("?"), Progress reads character-strings from the default text segment.
- The value of CURRENT-LANGUAGE might be a comma-separated list of language names. If so, Progress searches r-code for a text segment that matches the first language in the list. If that segment is not found, then it searches for a text segment for the next entry in the list until a segment is found.

- You can initialize the CURRENT-LANGUAGE variable with the Language (-lng) parameter.
- The behavior of CURRENT-LANGUAGE when one procedure calls another is as follows:
  - If a procedure changes the value of CURRENT-LANGUAGE, calls from the procedure to the CURRENT-LANGUAGE function return the name of the new language, but the procedure continues to use the character strings of the original language.
  - If the procedure then runs another procedure, when the called procedure gets control, calls from the called procedure to the CURRENT-LANGUAGE function return the name of the new language, and the called procedure uses the character strings of the new language.
  - When the called procedure finishes and control returns to the original procedure, calls from the original procedure to the CURRENT-LANGUAGE function return the name of the new language, but the original procedure continues to use the character strings of the original language.

See also COMPILE statement, CURRENT-LANGUAGE statement

CURRENT-LANGUAGE statement

Sets the CURRENT-LANGUAGE variable for the current OpenEdge session.

Note: Does not apply to SpeedScript programming.

Syntax

```
CURRENT-LANGUAGE = string-expression
```

string-expression

A character-string expression that specifies a language name or a comma-separated list of language names.

Example

This example procedure uses the CURRENT-LANGUAGE function to find the current language, prompts the user to choose a new language, and then uses the CURRENT-LANGUAGE statement to reset and display the name of the new current language:

r-chglng.p

```
DEFINE VARIABLE cur-lang AS CHARACTER FORMAT "x(10)" VIEW-AS RADIO-SET
    RADIO-BUTTONS czech,   "Czech",
                  danish,  "Danish",
                  dutch,   "Dutch",
                  english, "English",
                  french,  "French",
                  german,  "German",
                  hungar,  "Hungarian",
                  italian, "Italian",
                  norweg,  "Norwegian",
                  polish,  "Polish",
                  portug,  "Portuguese",
                  swedish, "Swedish".

IF CURRENT-LANGUAGE = "?"
THEN cur-lang = "English".
ELSE cur-lang = CURRENT-LANGUAGE.

UPDATE cur-lang NO-LABELS.

CURRENT-LANGUAGE = cur-lang.
MESSAGE "New language is" CURRENT-LANGUAGE.
```

Notes

- The value of CURRENT-LANGUAGE might be a comma-separated list of language names. If so, Progress searches r-code for a text segment that matches the first language in the list. If that segment is not found, then it searches for a text segment for the next entry in the list until a segment is found.
- You can initialize the CURRENT-LANGUAGE variable with the Language (-lng) parameter.
- The behavior of CURRENT-LANGUAGE when one procedure calls another is as follows:
  - If a procedure changes the value of CURRENT-LANGUAGE, calls from the procedure to the CURRENT-LANGUAGE function return the name of the new language, but the procedure continues to use the character strings of the original language.
  - If the procedure then runs another procedure, when the called procedure gets control, calls from the called procedure to the CURRENT-LANGUAGE function return the name of the new language, and the called procedure uses the character strings of the new language.
  - When the called procedure finishes and control returns to the original procedure, calls from the original procedure to the CURRENT-LANGUAGE function return the name of the new language, but the original procedure continues to use the character strings of the original language.

See also COMPILE statement, CURRENT-LANGUAGE statement

CURRENT-RESULT-ROW function

Returns the number of the current row of a specified query.

Syntax

```
CURRENT-RESULT-ROW ( query-name )
```

query-name

A character expression that evaluates to the name of a currently open, scrolling query. If *query-name* does not resolve to the name of a query, or if the query is not open or not scrolling, then the function returns the Unknown value (?).

Note: Searching for a query using a handle is more efficient than a character expression. Progress resolves a character expression at runtime by searching in the current routine for a static query with that name. If not found, Progress searches the enclosing main procedure. If still not found, Progress searches up through the calling programs of the current routine, and their main procedures. Since a handle uniquely identifies a query, no such search is required. Use the query object handle's CURRENT-RESULT-ROW attribute to avoid a runtime search.

Examples

The following example uses the QUERY-OFF-END function to determine when to leave the REPEAT loop:

r-resrow.p

```
DEFINE QUERY cust-query FOR customer SCROLLING.
OPEN QUERY cust-query FOR EACH customer WHERE Country = "USA".

REPEAT:
    GET NEXT cust-query.
    IF QUERY-OFF-END("cust-query") THEN LEAVE.
    DISPLAY CURRENT-RESULT-ROW("cust-query") LABEL "Result Row"
        Cust-num Name.
END.
```

Notes

- To use the CURRENT-RESULT-ROW function with a query, the query must be associated with a browse widget or you must define the query with the SCROLLING option. For more information on query definitions, see the reference entry for the DEFINE QUERY statement.
- If the query is empty, CURRENT-RESULT-ROW returns the Unknown value (?).
- If the query is positioned before the first record, CURRENT-RESULT-ROW returns the value 1. If the query is positioned beyond the last record, CURRENT-RESULT-ROW returns a value 1 greater than the number of rows in the query result list.
- When possible, Progress performs optimizations for GET LAST and REPOSITION statements. These optimizations make the results list invalid. At that point, CURRENT-RESULT-ROW returns the Unknown value (?). These optimizations do not occur if the query is opened with the PRESELECT option or has an associated browse widget.

See also

CLOSE QUERY statement, CURRENT-RESULT-ROW attribute, DEFINE BROWSE statement,DEFINE QUERY statement, GET statement, NUM-RESULTS function, OPEN QUERY statement, REPOSITION statement

CURRENT-VALUE function

Returns the current integer value of a sequence defined in the Data Dictionary.

Syntax

```
CURRENT-VALUE ( sequence [ , logical-dbname ] )
```

sequence

An identifier that specifies the name of a sequence defined in the Data Dictionary.

logical-dbname

An identifier that specifies the logical name of the database in which the sequence is defined. The database must be connected. You can omit this parameter if the sequence name is unambiguous. If a sequence with this name exists in more than one connected database, then you must specify *logical-dbname*.

Example The following example finds the current value of the next-cust-num sequence and then looks for orders with that customer number:

r-curval.p

```
DEFINE VARIABLE cur-cust LIKE customer.cust-num NO-UNDO.

cur-cust = CURRENT-VALUE(next-cust-num).
IF CAN-FIND(FIRST order WHERE order.cust-num = cur-cust) THEN
  FOR EACH order WHERE order.cust-num = cur-cust,
      EACH order-line OF order NO-LOCK
      BREAK BY order.order-num:

      IF FIRST-OF(order.order-num) THEN
        DISPLAY order.order-num order.order-date order.cust-num
          WITH FRAME order-info CENTERED ROW 2 1 COL.

      DISPLAY order-line.
  END.
ELSE DO:
  FIND FIRST customer WHERE customer.cust-num = cur-cust
        NO-LOCK NO-ERROR.

  IF AVAILABLE customer THEN
     MESSAGE "No Orders Exist for Customer " + customer.name +
           ", " + string(customer.cust-num)
           VIEW-AS ALERT-BOX INFORMATION BUTTONS OK TITLE "No Orders".
  ELSE MESSAGE "Customer number" cur-cust "does not exist."
           VIEW-AS ALERT-BOX INFORMATION BUTTONS OK TITLE "No Customer".
END.
```

Notes

- The current value of a sequence can be one of the following:
  - The initial value specified in the Data Dictionary.
  - The last value set with either the CURRENT-VALUE statement or the NEXT-VALUE function.
  - The Unknown value (?) if the sequence has exceeded its minimum or maximum and is not cycling.
- Sequence values are stored in the database in which they are defined, and persist between each invocation of the CURRENT-VALUE statement or NEXT-VALUE function.
- You cannot invoke the CURRENT-VALUE function from within a WHERE clause. Doing so generates a compiler error. To use a result from the CURRENT-VALUE function in a WHERE clause, assign the result to a variable, then use the variable in the WHERE clause.
- You can use any combination of the NEXT-VALUE function, CURRENT-VALUE function, CURRENT-VALUE statement, and their dynamic versions. Use the dynamic version when you don't know what the database name or sequence name is at runtime.

See also

CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

CURRENT-VALUE statement

Resets the current integer value of a sequence defined in the Data Dictionary.

Syntax

```
CURRENT-VALUE ( sequence [ , logical-dbname ] ) = expression
```

sequence

An identifier that specifies the name of a sequence defined in the Data Dictionary.

logical-dbname

An identifier that specifies the logical name of the database in which the sequence is defined. The database must be connected. You can omit this parameter if the sequence name is unambiguous. If more than one connected database has a sequence with given name, then you must supply *logical-dbname*.

expression

An integer expression assigned as the current value of the specified sequence. If *expression* is outside the boundary set by the initial value (at one end) and the lower limit or upper limit (at the other end) for the sequence, Progress returns an error, and the sequence value remains unchanged.

Example

The following example resets the current value of the next-cust-num sequence to the cust-num value of the last customer record if that is a valid value for the sequence:

r-curvl1.p

```
FIND LAST customer NO-LOCK.

IF customer.cust-num < CURRENT-VALUE(next-cust-num) AND
   customer.cust-num > 1000
THEN DO:
   CURRENT-VALUE(next-cust-num) = customer.cust-num.
   MESSAGE "The value of next-cust-num has been changed to"
           customer.cust-num VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
END.
ELSE MESSAGE "The value of next-cust-num remains"
             CURRENT-VALUE(next-cust-num)
             VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
```

Notes

- The user must have CAN-WRITE privileges on the _Sequence table to use the CURRENT-VALUE statement.
- The value of a sequence set by the CURRENT-VALUE statement persists in the database until the next CURRENT-VALUE statement or NEXT-VALUE function is invoked for the sequence, or until the sequence is deleted from the database.
- You cannot set a sequence to the Unknown value (?).
- You can use any combination of the NEXT-VALUE function, CURRENT-VALUE function, CURRENT-VALUE statement, and their dynamic versions. Use the dynamic version when you don't know what the database name or sequence name is at runtime.

See also

CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

Data types

The data type of a field defines what kind of data the field can store. All data types other than the MEMPTR, BLOB, CLOB, and LONGCHAR data types are limited in size to 32K. MEMPTR and LONGCHAR variables can be any size, and BLOB and CLOB fields can be up to 1GB in size.

Table 19 lists the data types supported by the Progress 4GL.

Table 19: Progress data types *(1 of 2)*

Data type	Description
BLOB	A BLOB (Binary Large OBject) consists of a database table or temp-table field that contains a BLOB locator, which points to the associated BLOB data stored in the database. You use a MEMPTR to manipulate the binary contents of a BLOB field in the 4GL.
CHARACTER	CHARACTER data consists of numbers, letters, and special characters.
CLASS	A user-defined class provides a common set of data members and methods for any object instance of that class. You can use a class just as you would any Progress built-in data type. You can define variables, parameters, and return types as a class. You can also assign an object reference for a class object instance to a temporary table field defined as a Progress.Lang.Object. For more information about defining a class, see the CLASS statement reference entry in this book. For information about Progress built-in classes, see the "Class Reference" section on page 2203.
CLOB	A CLOB (Character Large OBject) consists of a database table or temp-table field that contains a CLOB locator, which points to the associated CLOB data stored in the database. You use a LONGCHAR to manipulate the character contents of a CLOB field in the 4GL.
COM-HANDLE	A COM-HANDLE is a handle to a COM object (ActiveX Automation object or ActiveX Control).
DATE	DATE fields contain dates.
DATETIME	DATETIME data has two parts: a 4GL date and a 4GL time. The unit of time is milliseconds from midnight.

Table 19: Progress data types *(2 of 2)*

Data type	Description
DATETIME-TZ	DATETIME-TZ data has three parts: a 4GL date, a 4GL time, and an integer representing the time zone offset from Coordinated Universal Time (UTC). The unit of time is milliseconds from midnight.
DECIMAL	DECIMAL data consists of decimal numbers up to 50 digits in length including up to 10 digits to the right of the decimal point.
HANDLE	A HANDLE is a pointer to a Progress object. **Note:** HANDLE and WIDGET-HANDLE can be assigned to each other and used interchangeably.
INTEGER	INTEGER data consists of whole numbers.
LOGICAL	LOGICAL data evaluates to TRUE or FALSE (or YES or NO).
LONGCHAR	A LONGCHAR consists of CHARACTER data that is not limited to 32K in size. You can use a LONGCHAR to manipulate the character contents of a CLOB field in the 4GL.
MEMPTR	A MEMPTR contains a sequence of bytes in memory.
RAW	RAW data can be any kind of data, even data from non-OpenEdge databases. It is not converted in any way.
RECID	A RECID is an unique internal identifier of the current database record. **Note:** RECID is supported mainly for backward compatibility. For most applications, use ROWID instead.
ROWID	A ROWID is an unique internal identifier of the current database record.
WIDGET-HANDLE	A WIDGET-HANDLE is a pointer to a Progress widget. **Note:** HANDLE and WIDGET-HANDLE can be assigned to each other and used interchangeably.

Table 20 lists the default data formats for the data types.

Table 20: Default initial values and display formats *(1 of 2)*

Data type	Default initial value	Default display format
BLOB[1]	Unknown value (?)	See the footnote at the end of this table.
CHARACTER	"" (an empty string)	X(8)
CLASS[3]	Unknown value (?)	>>>>>>9
CLOB[1]	Unknown value (?)	See the footnote at the end of this table.
COM-HANDLE	Unknown value (?)	>>>>>>9
DATE	Unknown value (?) (displays as blanks)	99/99/99
DATETIME	Unknown value (?)	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	Unknown value (?)	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	0	->>,>>9.99
HANDLE[2]	Unknown value (?)	>>>>>>9
INTEGER	0	->,>>>,>>9
LOGICAL	no	yes/no
LONGCHAR[1]	"" (an empty string)	See the footnote at the end of this table.
MEMPTR[1]	A zero-length sequence of bytes.	See the footnote at the end of this table.
RAW[1]	A zero-length sequence of bytes.	See the footnote at the end of this table.

Table 20: Default initial values and display formats *(2 of 2)*

Data type	Default initial value	Default display format
RECID	Unknown value (?)	>>>>>>9
ROWID[1]	A zero-length sequence of bytes.	See the footnote at the end of this table.
WIDGET-HANDLE[2]	Unknown value (?)	>>>>>>9

1 You cannot display a BLOB, CLOB, MEMPTR, RAW, or ROWID value directly. However, you can convert a MEMPTR, RAW, or ROWID value to a character string representation using the STRING function and display the result. You can also convert a BLOB to a MEMPTR, and then use the STRING function. A MEMPTR or RAW value converts to a decimal integer string. A ROWID value converts to a hexadecimal string, "0x*hexdigits*," where *hexdigits* is any number of characters "0" through "9" and "A" through "F". You can display a CLOB field by converting it to a LONGCHAR, and displaying the LONGCHAR using the VIEW-AS EDITOR LARGE phrase only.

2 To display a HANDLE or WIDGET-HANDLE, you must first convert it using the INTEGER function and display the result.

3 To display a CLASS, you must first convert it using the INTEGER or STRING function and display the result.

For more information on using the different data types, see *OpenEdge Development: Progress 4GL Handbook*.

Notes

- In Version 9.0, when you copy one MEMPTR (M1) to another MEMPTR (M2), only the MEMPTR address is copied and both MEMPTRs point to the same memory location (L1). You can change the data in the single memory location and both MEMPTRs will point to the changed data. To clear memory after using the MEMPTRs, you can SET-SIZE = 0 on just one of the MEMPTRs.

  In Version 9.1, when you copy one MEMPTR (M1) to another MEMPTR (M2), the data that M1 points to is also copied. Therefore, MEMPTR M1 points to memory location L1, and MEMPTR M2 now points to memory location L2 which contains a copy of the data in L1. You must change the data in both memory locations if you want both MEMPTRs to reflect the change. To clear memory after using the MEMPTRs, you must execute SET-SIZE = 0 on both MEMPTRs to be sure that both memory locations are cleared.

- Starting with Version 9.1, you can assign RAW values to MEMPTR variables and MEMPTR values to RAW variables using the existing Progress assignment operator (=). If the target variable is a RAW data type, Progress will re-size the target variable, if necessary, so that after the assignment it will be the same size as the source. Note that after the assignment (whether RAW = MEMPTR or MEMPTR = RAW), the target variable has a copy of the memory associated with the source — each variable has an independent copy of the data.

- Since RAW variables are limited in size to 32K and MEMPTR variables are not limited in size, if a MEMPTR with a size greater than 32K is copied to a RAW variable, Progress will generate an error.

See also = Assignment operator, ASSIGN statement

DATASERVERS function

Returns a list of database types your OpenEdge product supports from where it is executed. The DATASERVERS function takes no arguments.

Syntax

```
DATASERVERS
```

The DATASERVERS function returns a character string containing a comma-separated list of database types. For example:

```
"PROGRESS,ODBC,ORACLE"
```

You can use the returned string with the LOOKUP function to determine whether a particular type of database is supported.

Example

The following example displays a selection list of all supported database types:

r-dserv.p

```
DEFINE VARIABLE db-types AS CHARACTER VIEW-AS SELECTION-LIST
   INNER-CHARS 20 INNER-LINES 3 LABEL "DataServers".
FORM
   db-types.
   db-types:LIST-ITEMS = DATASERVERS.
UPDATE db-types.
```

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, LOOKUP function, NUM-DBS function, PDBNAME function, SDBNAME function

DATA-SOURCE-MODIFIED function

Returns TRUE if data in the data source associated with the specified ProDataSet temp-table buffer has been modified.

Syntax

```
DATA-SOURCE-MODIFIED( buffer-name )
```

buffer-name

The name of a ProDataSet temp-table buffer.

Notes

- Progress sets this attribute from the SAVE-ROW-CHANGES( ) method.
- The DATA-SOURCE-MODIFIED function corresponds to the DATA-SOURCE-MODIFIED attribute.
- You can invoke the DATA-SOURCE-MODIFIED function from within a WHERE clause (unlike the corresponding attribute).

DATE function

Converts a single character string, a set of month, day, and year values, an integer expression, a DATETIME expression, or a DATETIME-TZ expression into a date.

If the DATE function cannot produce a valid date given the specified argument(s), it returns a runtime error.

Syntax

```
DATE ( month , day , year )
```

```
DATE ( string )
```

```
DATE ( integer-expression )
```

```
DATE ( datetime-expression )
```

month

A constant, field name, variable name, or expression whose value is an integer from 1 to 12, inclusive.

day

An expression whose value is an integer from 1 to the highest valid day of the *month*.

year

An expression whose value is the year (for example, 1994).

string

A character string containing a date value to convert into a DATE data type. The string value must have the format specified by the Date Format (-d) startup parameter (the default is mdy). Note that -d sets the display format, not the date storage format, which is fixed. Furthermore, date constants entered in procedures, or as initial values in the Data Dictionary, are always specified in month/day/year format.

You do not have to specify separator characters for the month, day, and year components of the date string; however, slashes(/), periods(.), and hyphens(-) are accepted as separator characters.

integer-expression

An expression that evaluates to a signed integer value that represents the number of days since the origin of the 4GL date data type. Usually this integer is obtained from a previous operation where the date was converted to an integer using the INTEGER(*4GL-date*) function.

The value of the expression cannot exceed the maximum date value, which is 12/31/32767.

Note: The resulting date from the DATE(*integer-expression*) function is guaranteed to be a valid 4GL date only if the *integer-expression* originated from the INTEGER(*4GL-date*) function.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The DATE function returns the date portion of the *datetime-expression* as a DATE.

If *datetime-expression* is a DATETIME-TZ, the DATE function returns the date relative to the time zone of the DATETIME-TZ value. For example, a DATETIME-TZ field, *fdt*, is created in London (time zone UTC+00:00) with a value of *May 5, 2002 at 7:15:03.002 am*. DATE (*fdt*) returns *05/05/2002*, regardless of the session's time zone.

Examples

This procedure reads data from an input file that contains date information from another system stored as character strings without slashes or dashes between month, day, and year. It tries to convert these dates to Progress dates. Some formats cannot be successfully converted. For example:

r-date.p

```
/* r-date.p */
DEFINE VARIABLE cnum as CHAR FORMAT "x(3)".
DEFINE VARIABLE cdate as CHAR FORMAT "x(16)".
DEFINE VARIABLE iday as INTEGER.
DEFINE VARIABLE imon as INTEGER.
DEFINE VARIABLE iyr as INTEGER.
DEFINE VARIABLE ddate as DATE.

INPUT FROM VALUE(SEARCH("r-date.dat")).
REPEAT:
  SET cnum cdate.
  imon = INTEGER(SUBSTR(cdate,1,2)).
  iday = INTEGER(SUBSTR(cdate,4,2)).
  iyr  = INTEGER(SUBSTR(cdate,7,2)).
  IF (iyr < 50)           /*WORKS FOR YEARS WITHIN 50 of 2000*/
  THEN
      iyr = iyr + 2000.
  ELSE
      iyr = iyr + 1900.
  ddate = DATE(imon,iday,iyr).
  DISPLAY ddate.
END.
INPUT CLOSE.
```

The following example shows the DATE (*string*) syntax:

r-date2.p

```
/* r-date2.p */

DEFINE VARIABLE cnum AS CHARACTER FORMAT "x(3)".
DEFINE VARIABLE cdate AS CHARACTER FORMAT "x(16)".
DEFINE VARIABLE ddate AS DATE FORMAT "99/99/9999".

INPUT FROM VALUE(SEARCH("r-date.dat")).

REPEAT:
   SET cnum cdate.
   ddate = DATE(cdate).
   DISPLAY ddate.
END.

INPUT CLOSE.
```

This example produces the following output. It produces no date for the first example since spaces are not a valid date separator:

```
cnum  cdate            ddate
----  ---------------- ----------

 nyd  01 01 86
 ad   11-28-52         11/28/1952
 red  12/08/56         12/08/1956
 nsm  10.01.01         10/01/2001
 hrl  1/8/1994         10/08/1994
```

See also DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

DATETIME function

Converts date and time values, or a character string, into a DATETIME.

Note: If any argument is the Unknown value (?), the result is the Unknown value (?).

Syntax

```
DATETIME (date-exp [, mtime-exp ] )
```

```
DATETIME ( string )
```

```
DATETIME (month, day, year, hours, minutes [, seconds [, milliseconds ] ] )
```

date-exp

An expression whose value is a DATE.

mtime-exp

An expression whose value is an integer representing the number of milliseconds since midnight.

string

A character expression whose value is a string containing a DATETIME. The date portion of the string must have the format specified by the DATE-FORMAT attribute. The time portion must be in a valid time format (HH:MM:SS, and so on).

month

An expression whose value is an integer from 1 to 12, inclusive.

day

An expression whose value is an integer from 1 to the highest valid day of the month.

year

An expression that evaluates to a year.

hours

An expression whose value is an integer from 0 to 23, inclusive.

minutes

An expression whose value is an integer from 0 to 59, inclusive.

seconds

An expression whose value is an integer from 0 to 61, inclusive. The upper limit is 61 for leap seconds.

milliseconds

An expression whose value is an integer from 0 to 999, inclusive.

Example

Following is an example of using the DATETIME function:

```
DEF VAR my-datetime as DATETIME.

my-datetime = DATETIME(TODAY, MTIME).

/* The statement above is equivalent to "my-datetime = NOW". */
```

The following statements result in the same DATETIME value:

```
my-datetime = DATETIME(5, 5, 2002, 7, 15, 3).
my-datetime = DATETIME("05-05-2002 07:15:03").
```

See also

DATE function, DATE-FORMAT attribute, DATETIME-TZ function, DAY function, ETIME function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

DATETIME-TZ function

Converts a date, time, and time zone value, or a character string, into a DATETIME-TZ.

Note: If any argument is the Unknown value (?), the result is the Unknown value (?).

Syntax

```
DATETIME-TZ (date-exp [, mtime-exp [, timezone-exp ] ] )
```

```
DATETIME-TZ (datetime-exp [, timezone-exp ] )
```

```
DATETIME-TZ (datetime-tz-exp [, timezone-exp ] )
```

```
DATETIME-TZ (month, day, year, hours, minutes [, seconds [, milliseconds
   [, timezone-exp ] ] ] )
```

```
DATETIME-TZ ( string )
```

date-exp

An expression whose value is a DATE.

mtime-exp

An expression whose value is an integer representing the number of milliseconds since midnight.

timezone-exp

An expression whose value is an integer representing the time zone offset from Coordinated Universal Time (UTC) in minutes. If not specified, the function uses the session's time zone.

datetime-exp

An expression whose value is a DATETIME.

datetime-tz-exp

An expression whose value is a DATETIME-TZ. Use this option to convert a DATETIME-TZ from one time zone to another.

month

An expression whose value is an integer from 1 to 12, inclusive.

day

An expression whose value is an integer from 1 to the highest valid day of the month.

year

An expression that evaluates to a year.

hours

An expression whose value is an integer from 0 to 23, inclusive.

minutes

An expression whose value is an integer from 0 to 59, inclusive.

seconds

An expression whose value is an integer from 0 to 61, inclusive. The upper limit is 61 for leap seconds.

milliseconds

An expression whose value is an integer from 0 to 999, inclusive.

string

A character expression whose value is a string containing a DATETIME-TZ. The date portion of the string must have the format specified by the DATE-FORMAT attribute. The time portion must be in a valid time format (HH:MM:SS, and so on). If the string contains a time zone, it must be in +HH:MM format. If the string does not contain a time zone, the DATETIME-TZ inherits the session's time zone.

Example Following is an example of using the DATETIME-TZ function:

```
DEF VAR my-datetime-tz as DATETIME-TZ.

my-datetime-tz = DATETIME-TZ(TODAY, MTIME, TIMEZONE).

/* The statement above is equivalent to "my-datetime-tz = NOW". */
```

The following statements result in the same DATETIME-TZ value (when SESSION:DATE-FORMAT is `mdy`):

```
my-datetime-tz = DATETIME-TZ(5, 5, 2002, 7, 15, 3, 0, -300).
my-datetime-tz = DATETIME-TZ("05-05-2002 07:15:03-05:00").
```

See also DATE function, DATE-FORMAT attribute, DATETIME function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function

DAY function

Evaluates a date expression and returns a day of the month integer value from 1 to 31, inclusive.

Syntax

```
DAY ( date )
```

```
DAY ( datetime-expression )
```

date

An expression whose value is a DATE.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The DAY function returns the day of the month of the date part of the DATETIME or DATETIME-TZ value.

Example

This procedure determines the date one year from a given date, allowing for leap years. You could simply determine a date 365 days later by adding 365 to the d1 variable, but that might not produce the correct result (for example, 1/1/92 + 365 days is 12/31/92).

r-day.p

```
DEFINE VARIABLE d1 AS DATE LABEL "Date".
DEFINE VARIABLE d2 AS DATE LABEL "Same date next year".
DEFINE VARIABLE d-day AS INTEGER.
DEFINE VARIABLE d-mon AS INTEGER.

REPEAT:
  SET d1.
  d-day = DAY(d1).
  d-mon = MONTH(d1).
  IF d-mon = 2 AND d-day = 29 THEN d-day = 28.
  d2 = DATE(d-mon,d-day,YEAR(d1) + 1).
  DISPLAY d2.
END.
```

See also

DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, ETIME function, ISO-DATE function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

DBCODEPAGE function

Returns, as a character string, the name of a connected database's code page.

Syntax

```
DBCODEPAGE ( { integer-expression | logical-name | alias } )
```

integer-expression

The sequence number of a database the OpenEdge session is connected to. For example, DBCODEPAGE(1) returns information on the first database the OpenEdge session is connected to, DCODEPAGE(2) returns information on the second database the OpenEdge session is connected to, etc. If you specify a sequence number that does not correspond to a database the OpenEdge session is connected to, the DBCODEPAGE function returns the Unknown value (?).

logical-name

A character expression that specifies the database by its logical name or alias.

A code page maps each character in a character set to a numeric value. For an OpenEdge database, DBCODEPAGE returns the code page of the database represented by the integer expression, logical name, or alias. For a non-OpenEdge database, DBCODEPAGE returns the value originally inserted when the schema was created.

There are three possible types of non-OpenEdge code pages:

- Physical data source for the database.
- Code page of a non-OpenEdge vendor library linked in with an OpenEdge dataserver executable (either dynamically or statically).
- Code page that is in the schema holder that is part of the create activity.

If any parameter is invalid, it returns the Unknown value (?).

Example

This procedure displays the logical name and code page of all connected databases:

r-dbcp.p

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i=1 TO NUM-DBS:
  DISPLAY LDBNAME(i) DBCODEPAGE(i) FORMAT "x(19)".
END.
```

Note

A database must be connected in order for the DBCODEPAGE function to work as described.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, PDBNAME function, SDBNAME function

DBCOLLATION function

Returns, as a character string, the name of the collating sequence for character set information contained in the database. This name corresponds to the definition of the collating sequence contained in the `convmap.dat` file, which usually resides in the $DLC directory. If any parameter is invalid, DBCOLLATION returns the Unknown value (?).

Syntax

```
DBCOLLATION
  ( { integer-expression | logical-name | alias } )
```

integer-expression

The sequence number of a database the OpenEdge session is connected to. For example, DBCOLLATION(1) returns information on the first database the OpenEdge session is connected to, DBCOLLATION(2) returns information on the second database the OpenEdge session is connected to, etc. If you specify a sequence number that does not correspond to a database the OpenEdge session is connected to, the DBCOLLATION function returns the Unknown value (?).

logical-name or *alias*

A character expression that specifies the database by its logical name or alias.

Example

This procedure displays the logical name and collation of all connected databases:

r-dbcoll.p

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i=1 TO NUM-DBS:
  DISPLAY LDBNAME(i) DBCOLLATION(i) FORMAT "x(19)".
END.
```

Notes

- OpenEdge and non-OpenEdge dataservers can evaluate the syntactical expression stated in a DBCOLLATION function. However, the methods used to process multiple byte code pages can differ based on the actual server used. Keep this point in mind if the actual results you receive differ from the results you expected.
- A database must be connected in order for the DBCOLLATION function to work as described.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBRESTRICTIONS function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DBNAME function

Returns, as a character string, the name of the logical database currently in use or the name of your first connected database.

Syntax

```
DBNAME
```

Example

This portion of a procedure defines a header frame to hold a date, page number, database name, and user ID:

r-dbname.p

```
DEFINE VARIABLE pageno AS INTEGER FORMAT "zzz9" INITIAL 1.

FORM HEADER "Date:" TO 10 TODAY
    "Page:" AT 65 pageno SKIP
    "Database:" TO 10 DBNAME FORMAT "x(60)" SKIP
    "Userid:" TO 10 userid WITH NO-BOX NO-LABELS.
VIEW.
```

Notes

- Progress returns the database name in the same form you used when you connected to the database. If you used a fully qualified pathname, Progress returns the full directory pathname (such as `/usr/acctg/gl` on UNIX or `\acctg\gl` in Windows). If you used a name relative to your working directory, then Progress returns that name (for example, gl).
- Unless you define a format, the database name is displayed in a character field with the default format of x(8).
- A database must be connected in order for the DBNAME function to work as described.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DBPARAM function

Returns, as a character string, a comma-separated list of the parameters used to connect to the database.

Syntax

```
DBPARAM ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a database the OpenEdge session is connected to. For example, DBPARAM(1) returns information on the first database the OpenEdge session is connected to, DBPARAM(2) returns information on the second database the OpenEdge session is connected to, etc. If you specify a sequence number that does not correspond to a database the OpenEdge session is connected to, the DBPARAM function returns the Unknown value (?).

logical-name or *alias*

These forms of the DBPARAM function require a character expression as a parameter. An unquoted character string is not permitted. If the parameter is an alias or the logical name of a connected database, then Progress returns the comma-separated parameter list. Otherwise, it returns the Unknown value (?).

Notes

- A database must be connected for the DBPARAM function to work as described.
- If the CONNECT statement does not contain a -db (database) parameter, which is permissible, the string DBPARAM returns includes the -db parameter and the database name.
- If the CONNECT statement contains the -pf parameter, which refers to a parameter file, the string DBPARAM returns includes the parameters in the file without "-pf" or any reference to the file.
- If the CONNECT statement contains a userid and a password, the string DBPARAM returns includes only the userid.
- The database can connect through the CONNECT statement, the command line, or an auto-connection.

See also DBCODEPAGE function, DBCOLLATION function, DBTYPE function, DBVERSION function

DBRESTRICTIONS function

Returns a character string that describes features that are not supported for this database. You can use this function with OpenEdge DataServers.

Syntax

```
DBRESTRICTIONS
  ( { integer-expression | logical-name | alias }
    [ , table-name ]
  )
```

integer-expression

The sequence number of a database the OpenEdge session is connected to. For example, DBRESTRICTIONS(1) returns information on the first database the OpenEdge session is connected to, DBRESTRICTIONS(2) returns information on the second database the OpenEdge session is connected to, and so on. If you specify a sequence number that does not correspond to a database the OpenEdge session is connected to, the DBRESTRICTIONS function returns the Unknown value (?).

logical-name or *alias*

These forms of the DBRESTRICTIONS function require a character expression as a parameter. An unquoted character string is not permitted. If the parameter is an alias or the logical name of a connected database, then Progress returns the database restrictions string. Otherwise, it returns the Unknown value (?).

table-name

A character expression equal to the name of a table in the specified database. An unquoted character string is not permitted. If the table name is valid, DBRESTRICTIONS returns the list of unsupported features for the specified table. Otherwise, it returns the Unknown value (?).

Example This procedure displays the logical name and database restrictions of all connected databases:

r-dbrest.p

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i= 1 to NUM-DBS:
    DISPLAY LDBNAME(i) LABEL "Database"
            DBRESTRICTIONS(i) FORMAT "x(40)" LABEL "Restrictions".
END.
```

Notes

- If you want to use the DBRESTRICTIONS function for a database, you must be connected to the database in the current OpenEdge session.
- DBRESTRICTIONS returns a string. This string is a comma-separated list of keywords that represent features not supported by the specified database. Table 21 shows the possible keywords and their descriptions.

Table 21: DBRESTRICTIONS keyword values

Keyword	Description
COUNT-OF	Cannot use the COUNT-OF function.
LAST	Cannot invoke the FIND LAST statement.
PREV	Cannot invoke the FIND PREV statement.
READ-ONLY	The database or table is available for read only.
RECID	Cannot use the RECID function.
SET-CURRENT-VALUE	Cannot set the current value of sequence generators.
SETUSERID	Cannot use the SETUSERID function.

For example, if the database is accessed through a manager that does not support FIND LAST and FIND PREV, then the DBRESTRICTIONS function returns the string LAST, PREV.

- The possible keyword values returned by DBRESTRICTIONS depends on the DataServer type. Table 22 shows the possible values returned for each DataServer.

Table 22: DBRESTRICTIONS return values by DataServer

DataServer	Possible return values
ODBC	"COUNT-OF,LAST,PREV,READ-ONLY,RECID,SETUSERID"
ORACLE	"LAST,PREV,READ-ONLY,RECID,SETUSERID, SET-CURRENT-VALUE"
OpenEdge	"READ-ONLY"

Note: The available DataServers depend on your version of OpenEdge. For more information, see your OpenEdge DataServer documentation.

- The form of the returned string makes it easy to use with the ENTRY and LOOKUP function.
- If you connect to a database with the Read Only (-RO) parameter, Progress lists the character string READ-ONLY in the restrictions list for that database.

See also ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DBTASKID function

Returns an INTEGER that uniquely identifies a database's transaction.

Syntax

```
DBTASKID ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a database the OpenEdge session is connected to. For example, DBTASKID(1) returns information on the first database the OpenEdge session is connected to, DBTASKID(2) returns information on the second database the OpenEdge session is connected to, etc. If you specify a sequence number that does not correspond to a database the OpenEdge session is connected to, the DBTASKID function returns the Unknown value (?).

logical-name or *alias*

A character expression that evaluates to the logical name or alias of a database that is connected to the current OpenEdge session. If the character expression does not evaluate to the logical name or alias of a connected database, DBTASKID returns the Unknown value (?).

Note: You must enclose all character strings in quotes.

Notes

- If the application is not in a transaction, DBTASKID returns the Unknown value (?).
- If the client is connected to two databases and both databases participate in the transaction, DBTASKID does not necessarily return the same value for each database. The value DBTASKID returns for a database is for that database only.
- DBTASKID supports Version 8 and Version 9 Progress databases only. DBTASKID returns the Unknown value (?) for DataServers, Version 7 Progress databases, and the temporary table database.
- DBTASKID is designed for database replication. When you create a log record for a transaction, you can call DBTASKID and store the transaction ID. When you load the transaction, you can group log records by transaction ID. For more information on database replication, see *OpenEdge Data Management: Database Administration*, and the reference entry for the RAW-TRANSFER statement in this book.

See also

DBCODEPAGE function, DBCOLLATION function, DBTYPE function, DBVERSION function, LDBNAME function

DBTYPE function

Returns, as a character string, the database type of a currently connected database. Database types include the following: OpenEdge, ODBC, and ORACLE.

Syntax

```
DBTYPE ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a database the OpenEdge session is connected to. For example, DBTYPE(1) returns information on the first database the OpenEdge session is connected to, DBTYPE(2) returns information on the second database the OpenEdge session is connected to, etc. If you specify a sequence number that does not correspond to a database the OpenEdge session is connected to, the DBTYPE function returns the Unknown value (?).

logical-name or *alias*

These forms of the DBTYPE function require a quoted character string or a character expression as a parameter. An unquoted character string is not permitted. If the parameter is an alias of a connected database or the logical name of a connected database, then Progress returns the database type. Otherwise, it returns the Unknown value (?).

Example This procedure displays the logical name and database type of all connected databases:

r-dbtype.p

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i=1 TO NUM-DBS:
  DISPLAY LDBNAME(i) DBTYPE(i) FORMAT "x(40)".
END.
```

Note You can reference the DBTYPE function within a preprocessor expression. For more information, see the &IF, &THEN, &ELSEIF, &ELSE, and &ENDIF preprocessor directives reference entry.

See also ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DBVERSION function

Returns, as a character string, the version number of an OpenEdge database.

Syntax

```
DBVERSION ( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a database the OpenEdge session is connected to. For example, DBVERSION(1) returns information on the first database the OpenEdge session is connected to, DBVERSION(2) returns information on the second database the OpenEdge session is connected to, etc. If you specify a sequence number that does not correspond to a database the OpenEdge session is connected to, the DBVERSION function returns the Unknown value (?).

logical-name or *alias*

These forms of the DBVERSION function require a quoted character string or a character expression as a parameter. If the parameter is an alias of a connected database or the logical name of a connected database, then Progress returns the version number. Otherwise, it returns the Unknown value (?).

Example

This procedure displays the version number of all connected databases:

r-dbvers.p

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i=1 TO NUM-DBS:
   DISPLAY LDBNAME(i) DBVERSION(i) WITH 1 DOWN.
END.
```

Note

DBVERSION does not apply to non-OpenEdge data sources.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DDE ADVISE statement (Windows only)

Instructs the dynamic data exchange (DDE) server associated with a conversation to either create or remove an advise link to the specified data item.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE ADVISE ddeid { START | STOP } ITEM name
  [ TIME seconds ]
  [ NO-ERROR ]
```

ddeid

An integer expression equal to the channel number of the conversation opened for the specified data item. It is the value returned by the DDE INITIATE statement that opened the conversation.

START

Instructs the server to create a link to a data item, and notify the OpenEdge client when the specified data item changes value.

STOP

Instructs the server to remove the link to the specified data item, and stop monitoring its value.

ITEM *name*

Specifies the name of the server data item to which the link is created or removed. The data item *name* is a character expression that identifies the data item according to the conventions of the server application (for example, the row and column coordinates of a worksheet cell, such as R2C1 in Microsoft Excel). After creating a link, when the value of the data item specified by *name* changes, Progress triggers a DDE-NOTIFY event for the frame that owns the conversation, allowing the client to retrieve the new value.

TIME *seconds*

Specifies the maximum number of seconds that the OpenEdge client waits for the DDE ADVISE statement to complete, where *seconds* is an integer expression. If you do not specify the TIME option or specify a value of 0, Progress waits indefinitely for the statement to complete.

NO-ERROR

By default, if the statement fails to create or remove the link, it sets the Progress error condition, and posts the error to the DDE frame DDE-ERROR attribute. If you specify NO-ERROR, the statement does not set the Progress error condition, but does post the error to the DDE frame.

Example

The following fragment shows how to use the DDE ADVISE to set up a procedure to capture a rate-of-change value as it changes in a dynamic model run in a Microsoft Excel worksheet. The example assumes that the Microsoft Excel application is running, and has opened the default Excel worksheet, Sheet1, which runs the model.

After the conversation is opened, the DDE ADVISE statement links to the worksheet cell that maintains the latest rate-of-change value (second column of the fourth row, or R4C2). Every time this cell changes value, Progress posts a DDE-NOTIFY event to the frame DDEframe, where the value is retrieved using the DDE GET statement, and stored as a decimal in the ChangeRate variable. Meanwhile, if the REPEAT block detects a ChangeRate value greater than 7.5%, the link to cell R4C2 is closed and the procedure continues.

```
DEFINE VARIABLE Sheet1 AS INTEGER.         /* DDE-ID to worksheet     */
DEFINE VARIABLE ChangeRate AS DECIMAL      /* Rate of change variable */
  INITIAL 0.0.                            /*    starting at zero     */
DEFINE VARIABLE CellData AS CHARACTER.     /* Worksheet cell output   */
DEFINE VARIABLE DDEframe AS WIDGET-HANDLE. /* DDE frame handle        */

CREATE FRAME DDEframe
TRIGGERS: /* DDE frame and code to receive  */
  ON DDE-NOTIFY DO:      /* rate of change data from Excel */
    DDE GET Sheet1 TARGET CellData ITEM "R4C2".
    ChangeRate = DECIMAL(CellData).
  END.
END TRIGGERS.
                      .
                      .
                      .
/* Open conversation with "Sheet1" and link to rate-of-change value. */

DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE ADVISE Sheet1 START ITEM "R4C2".

/* Do some processing while the rate-of-change is within 7.5% */

REPEAT WHILE ChangeRate <= 7.5:
                      .
                      .
                      .
END. /* 7.5% processing */

/* Go on to other things once the rate of change goes above 7.5%. */

DDE ADVISE Sheet1 STOP ITEM "R4C2".
                      .
                      .
                      .
```

Notes

- After a DDE-NOTIFY event is triggered for the conversation, the client application must use the DDE GET statement in a trigger block for the event to retrieve the latest value for *name*.
- For more information on using the DDE protocol to exchange data with non-OpenEdge applications, see *OpenEdge Development: Programming Interfaces*.

See also DDE GET statement, DDE INITIATE statement

DDE EXECUTE statement (Windows only)

Instructs a dynamic data exchange (DDE) server application to execute one or more application commands.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE EXECUTE ddeid COMMAND string
  [ TIME seconds ]
  [ NO-ERROR ]
```

ddeid

An integer expression equal to the channel number of a conversation opened to execute the specified command string. It is the value returned by the DDE INITIATE statement that opened the conversation. You can usually execute commands using a conversation opened for the System topic of the server application.

COMMAND *string*

Specifies the command or commands for the server to execute, where *string* is a character expression containing commands that are defined by the server application (for example, the [select(...)] command in Microsoft Excel).

TIME *seconds*

Specifies the maximum number of seconds that the OpenEdge client waits for the DDE EXECUTE statement to complete, where *seconds* is an integer expression. If you do not specify the TIME option or specify a value of 0, Progress waits indefinitely for the statement to complete.

NO-ERROR

By default, if the statement fails to execute the command(s), it sets the Progress error condition and posts the error to the DDE frame DDE-ERROR attribute. If you specify NO-ERROR, the statement does not set the Progress error condition but does post the error to the DDE frame.

Example

The following fragment shows how to use the DDE EXECUTE statement. The procedure executes Microsoft Excel internally and opens a conversation for the Excel System topic. The System topic lets you execute Excel functions. This example uses the DDE EXECUTE statement to create a new Excel worksheet using the NEW function:

```
DEFINE VARIABLE Sys AS INTEGER.            /* DDE-ID to System topic */
DEFINE VARIABLE DDEframe AS WIDGET-HANDLE. /* DDE frame handle       */
CREATE FRAME DDEframe.                     /* Create DDE frame.      */

/* DLL routine to execute an MS-Windows application. */

PROCEDURE WinExec EXTERNAL "kernel32.dll":
  DEFINE INPUT PARAMETER ProgramName AS CHARACTER.
  DEFINE INPUT PARAMETER Presentation AS LONG.
END PROCEDURE. /* WinExec */
             .
             .
             .
/* Start Excel, open a DDE conversation with the Excel System topic, */
/* and create a worksheet.                                           */

RUN WinExec (INPUT "Excel /e", INPUT 2). /* 1=normal, 2=minimized */
DDE INITIATE Sys FRAME DDEframe APPLICATION "Excel" TOPIC "System".
DDE EXECUTE Sys COMMAND "[new(1)]".
             .
             .
             .
```

Notes

- For more information on commands available in your server application, see the documentation for that application.
- For more information on using the DDE protocol to exchange data with non-OpenEdge applications, see *OpenEdge Development: Programming Interfaces*.

See also

DDE INITIATE statement

DDE GET statement (Windows only)

Retrieves the value of a dynamic data exchange (DDE) server data item that has changed and triggered a DDE-NOTIFY event.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE GET ddeid TARGET field ITEM name
  [ TIME seconds ]
  [ NO-ERROR ]
```

ddeid

An integer expression that specifies the channel number of the conversation that triggered the DDE-NOTIFY event. You can obtain the value of *ddeid* from the DDE-ID attribute of the frame to which the DDE-NOTIFY event was posted.

TARGET *field*

Specifies a character field or variable that receives the value of the server data item as a character string.

ITEM *name*

Specifies the server data item that changed and triggered the DDE-NOTIFY event, where *name* is a character expression that identifies the name of the data item in the server application. You can obtain the value of *name* from the DDE-ITEM attribute of the frame to which the DDE-NOTIFY event was posted.

`TIME` *seconds*

Specifies the maximum number of seconds that the OpenEdge client waits for the DDE GET statement to complete where *seconds* is an integer expression. If you do not specify the TIME option or specify a value of 0, Progress waits indefinitely for the statement to complete.

`NO-ERROR`

By default, if the statement fails to retrieve the data item value, it sets the Progress error condition, and posts the error to the DDE frame DDE-ERROR attribute. If you specify NO-ERROR, the statement does not set the Progress error condition but does post the error to the DDE frame.

Example

The following fragment shows how to use the DDE GET statement to set up a procedure to capture a rate-of-change value as it changes in a dynamic model run in a Microsoft Excel worksheet. The example assumes that the Microsoft Excel application is running, and has opened the default Excel worksheet, Sheet1, which runs the model.

After the conversation is opened, the DDE ADVISE statement links to the worksheet cell that maintains the latest rate-of-change value (2nd column of the 4th row, or R4C2). Every time this cell changes value, Progress posts a DDE-NOTIFY event to the frame DDEframe, where the value is retrieved using the DDE GET statement, and stored as a decimal in the `ChangeRate` variable. Meanwhile, if the REPEAT block detects a `ChangeRate` value greater than 7.5%, the the link to cell R4C2 is closed and the procedure continues.

```
DEFINE VARIABLE Sheet1 AS INTEGER.         /* DDE-ID to worksheet     */
DEFINE VARIABLE ChangeRate AS DECIMAL      /* Rate of change variable */
   INITIAL 0.0.                            /*    starting at 0     */
DEFINE VARIABLE CellData AS CHARACTER.     /* Worksheet cell output   */
DEFINE VARIABLE DDEframe AS WIDGET-HANDLE. /* DDE frame handle        */

CREATE FRAME DDEframe
TRIGGERS: /* DDE frame and code to receive  */
  ON DDE-NOTIFY DO:      /* rate of change data from Excel */
    DDE GET Sheet1 TARGET CellData ITEM "R4C2".
    ChangeRate = DECIMAL(CellData).
  END.
END TRIGGERS.
                        .
                        .
                        .
/* Open conversation with Sheet1 and link to rate-of-change value. */
DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE ADVISE Sheet1 START ITEM "R4C2".

/* Do some processing while the rate-of-change is within 7.5% */
REPEAT WHILE ChangeRate <= 7.5:
                        .
                        .
                        .
END. /* 7.5% processing */

/* Go on to other things once the rate of change goes above 7.5%. */


DDE ADVISE Sheet1 STOP ITEM "R4C2".
                        .
                        .
                        .
```

Notes

- Progress posts each DDE-NOTIFY event to the frame that owns the conversation opened for the linked data item.
- You can invoke this function in the trigger block for each frame that owns a conversation containing advise links. Only frames that own conversations linked to data items with the DDE-ADVISE statement can receive DDE-NOTIFY events.
- For more information on using the DDE protocol to exchange data with non-OpenEdge applications, see *OpenEdge Development: Programming Interfaces*.

See also DDE ADVISE statement, DDE INITIATE statement

DDE INITIATE statement (Windows only)

Opens a dynamic data exchange (DDE) client conversation for a specified DDE server application and topic, and associates the new conversation with a Progress frame. To identify the conversation, the statement returns an integer as a unique channel number for this conversation.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE INITIATE ddeid FRAME frame-handle
  APPLICATION server-name TOPIC topic-name [ NO-ERROR ]
```

ddeid-var

An integer variable or field that receives the channel number for the newly opened DDE conversation.

FRAME *frame-handle*

Specifies the handle of the frame that owns the conversation, where *frame-handle* is a widget handle expression. A frame can own more than one conversation. Progress records the status of the most recent conversation exchange in a set of DDE frame attributes. These attributes record the status of every dynamic data exchange, including advise exchanges (exchanges triggered by DDE-NOTIFY events). The DDE frame attributes include:

- **DDE-ERROR** — The DDE error code returned by the most recent exchange.
- **DDE-ID** — The channel number of the conversation that had the most recent exchange.
- **DDE-ITEM** — The name of the data item referenced by the most recent exchange.
- **DDE-NAME** — The name of the server application in the most recent exchange.
- **DDE-TOPIC** — The name of the topic of the most recent exchange.

APPLICATION *server-name*

Specifies the name of the server application for the conversation, where *server-name* is a character expression. The value of *server-name* must be unique for each DDE server on your system. It is usually the filename of the server executable without the extension (for example, the name EXCEL in Microsoft Excel).

TOPIC *topic-name*

Specifies the name of the topic of the conversation, where *topic-name* is a character expression. The value of *topic-name* identifies a category defined by the server application. This is usually the name of a file or other container that includes one or more data items (for example, the name of a worksheet, such as Sheet1 in Microsoft Excel). An OpenEdge client can only exchange data with server data items included in the topic of an open conversation.

NO-ERROR

By default, if the statement fails to open a conversation, it sets the Progress error condition and posts the error to the DDE frame DDE-ERROR attribute. If you specify NO-ERROR, the statement does not set the Progress error condition but does post the error to the DDE frame.

Example

The following fragment shows a typical use of the DDE INITIATE statement. It assumes that the Microsoft Excel application is running, and has created the default Excel worksheet, Sheet1. It then uses the DDE INITIATE statement to open a conversation with Sheet1 as the topic. This allows Progress to exchange data with the cells of the worksheet. In this example, the fragment assigns column headings to the top row of the first three columns in the worksheet:

```
DEFINE VARIABLE Sheet1 AS INTEGER.        /* DDE-ID to worksheet topic */
DEFINE VARIABLE DDEframe AS WIDGET-HANDLE. /* DDE frame handle         */

CREATE FRAME DDEframe.                     /* Create DDE frame.       */
                         .
                         .
                         .
/* Open a DDE conversation with Sheet1 and assign column headings. */

DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE SEND Sheet1 SOURCE "Name"      ITEM "R1C1".
DDE SEND Sheet1 SOURCE "YTD Sales" ITEM "R1C2".
DDE SEND Sheet1 SOURCE "State"     ITEM "R1C3".
                         .
                         .
                         .
```

Notes

- The specified DDE server application must be running on the Windows desktop before you can invoke the DDE INITIATE statement.
- You can close a DDE conversation in three ways: use the DDE TERMINATE statement, leave the scope of the frame that owns the conversation, or terminate the server application or topic associated with the application.
- For more information on using the DDE protocol (including DDE frame attributes) to exchange data with non-OpenEdge applications, see *OpenEdge Development: Programming Interfaces*.

See also DDE TERMINATE statement

DDE REQUEST statement (Windows only)

Retrieves the current value of a dynamic data exchange (DDE) server data item associated with the specified DDE conversation.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE REQUEST ddeid TARGET field ITEM name
  [ TIME seconds ]
  [ NO-ERROR ]
```

ddeid

An integer expression that equals the channel number of the conversation opened for the specified data item. It is the value returned by the DDE INITIATE statement that opened the conversation.

TARGET *field*

Specifies a character field or variable that receives the value of the data item as a character string.

ITEM *name*

Specifies the name of the server data item from which to retrieve a value. The data item *name* is a character expression that identifies the data item according to the conventions of the server application (for example, the row and column coordinates of a worksheet cell, such as R2C1 in Microsoft Excel).

TIME *seconds*

Specifies the maximum number of seconds that the OpenEdge client waits for the DDE REQUEST statement to complete, where *seconds* is an integer expression. If you do not specify the TIME option or specify a value of 0, Progress waits indefinitely for the statement to complete.

NO-ERROR

By default, if the statement fails to retrieve the data item value, it sets the Progress error condition and posts the error to the DDE frame DDE-ERROR attribute. If you specify NO-ERROR, the statement does not set the Progress error condition but does post the error to the DDE frame.

Example

The following fragment shows a typical use of the DDE REQUEST statement. It assumes that the Microsoft Excel application is running, and has created the default Excel worksheet, Sheet1. It then uses the DDE INITIATE statement to open a conversation with Sheet1 as the topic. This allows Progress to exchange data with the cells of the worksheet.

In this example, the fragment builds 10 new customer records from data obtained from the first 4 columns in the worksheet using the DDE REQUEST statement. The data includes customer name, year-to-date sales, state, and zip code. (The requests start from row 2, because row 1 contains column headings.)

```
DEFINE VARIABLE Rowi AS INTEGER.          /* Worksheet row counter     */
DEFINE VARIABLE ItemName AS CHARACTER.    /* Item Name                 */
DEFINE VARIABLE CustName AS CHARACTER.    /* Customer name receptor    */
DEFINE VARIABLE YTDsales AS CHARACTER.    /* YTD sales receptor        */
DEFINE VARIABLE StateAbr AS CHARACTER.    /* State name receptor       */
DEFINE VARIABLE ZipCode AS CHARACTER.     /* Zip code receptor         */
DEFINE VARIABLE Sheet1 AS INTEGER.        /* DDE-ID to worksheet topic */
DEFINE VARIABLE DDEframe AS WIDGET-HANDLE./* DDE frame handle          */

CREATE FRAME DDEframe.                    /* Create DDE frame.         */
                        .
                        .
                        .
/* Open a DDE conversation with Sheet1 and create 10 customer */
/* records from the data in four columns of the worksheet.      */

DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".

REPEAT Rowi = 2 TO 11:
  CREATE customer.
  customer.cust-num = Rowi - 1.
  ItemName = "R" + STRING(Rowi) + "C1".
  DDE REQUEST Sheet1 TARGET CustName ITEM ItemName.
  customer.name = CustName.
  ItemName = "R" + STRING(Rowi) + "C2".
  DDE REQUEST Sheet1 TARGET YTDsales ITEM ItemName.
  customer.ytd-sls = DECIMAL(YTDsales).
  ItemName = "R" + STRING(Rowi) + "C3".
  DDE REQUEST Sheet1 TARGET StateAbr ITEM ItemName.
  customer.st = StateAbr.
  ItemName = "R" + STRING(Rowi) + "C4".
  DDE REQUEST Sheet1 TARGET ZipCode  ITEM ItemName.
  customer.zip = INTEGER(ZipCode).
  RELEASE customer.
END.
                        .
                        .
                        .
```

Note For more information on using the DDE protocol to exchange data with non-OpenEdge applications, see *OpenEdge Development: Programming Interfaces*.

See also DDE INITIATE statement

DDE SEND statement (Windows only)

Sends a new value to a dynamic data exchange (DDE) server data item associated with the specified DDE conversation.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE SEND ddeid SOURCE data ITEM name
  [ TIME seconds ]
  [ NO-ERROR ]
```

ddeid

An integer expression that equals the channel number of the conversation opened for the specified data item. It is the value returned by the DDE INITIATE statement that opened the conversation.

SOURCE *data*

Specifies the new value for the server data item, where *data* is a character expression that renders the new value in a format acceptable to the data item.

ITEM *name*

Specifies the name of the server data item to receive the new value. The data item *name* is a character expression that identifies the data item according to the conventions of the server application (for example, the row and column coordinates of a worksheet cell, such as R2C1 in Microsoft Excel).

TIME *seconds*

Specifies the maximum number of seconds that the OpenEdge client waits for the DDE SEND statement to complete, where *seconds* is an integer expression. If you do not specify the TIME option or specify a value of 0, Progress waits indefinitely for the statement to complete.

NO-ERROR

By default, if the statement fails to send the value to the data item, it sets the Progress error condition and posts the error to the DDE frame DDE-ERROR attribute. If you specify NO-ERROR, the statement does not set the Progress error condition, but does post the error to the DDE frame.

Example

The following fragment shows a typical use of the DDE SEND statement. It assumes that the Microsoft Excel application is running, and has created the default Excel worksheet, Sheet1. It then uses the DDE INITIATE statement to open a conversation with Sheet1 as the topic. This allows Progress to exchange data with the cells of the worksheet. In this example, the fragment assigns column headings to the top row of the first three columns in the worksheet using the DDE SEND statement.

```
DEFINE VARIABLE Sheet1 AS INTEGER.        /* DDE-ID to worksheet topic */
DEFINE VARIABLE DDEframe AS WIDGET-HANDLE./* DDE frame handle          */

CREATE FRAME DDEframe.                    /* Create DDE frame.         */
                          .
                          .
                          .
/* Open a DDE conversation with Sheet1 and assign column headings. */

DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE SEND Sheet1 SOURCE "Name"      ITEM "R1C1".
DDE SEND Sheet1 SOURCE "YTD Sales" ITEM "R1C2".
DDE SEND Sheet1 SOURCE "State"     ITEM "R1C3".
                          .
                          .
                          .
```

Note

For more information on using the DDE protocol to exchange data with non-OpenEdge applications, see *OpenEdge Development: Programming Interfaces*.

See also

DDE INITIATE statement

DDE TERMINATE statement (Windows only)

Closes the specified dynamic data exchange (DDE) conversation.

Note: Does not apply to SpeedScript programming.

Syntax

```
DDE TERMINATE ddeid [ NO-ERROR ]
```

ddeid

An integer expression that equals the channel number of an open conversation. It is the value returned by the DDE INITIATE statement that opened the conversation.

NO-ERROR

By default, if the statement fails to close the conversation, it sets the Progress error condition, and posts the error to the DDE frame DDE-ERROR attribute. If you specify NO-ERROR, the statement does not set the Progress error condition but does post the error to the DDE frame.

Example

The following fragment shows a typical use of the DDE TERMINATE statement. It assumes that the Microsoft Excel application is running, and has created the default Excel worksheet, Sheet1. It then uses the DDE INITIATE statement to open a conversation with Sheet1 as the topic, returning the channel number of the conversation to the variable, Sheet1. After exchanging data with the worksheet, the example closes the conversation with Sheet1 using the DDE TERMINATE statement.

```
DEFINE VARIABLE Sheet1 AS INTEGER.        /* DDE-ID to worksheet topic */
DEFINE VARIABLE DDEframe AS WIDGET-HANDLE./* DDE frame handle          */

CREATE FRAME DDEframe.                    /* Create DDE frame.         */
                         .
                         .
                         .
/* Open a DDE conversation with "Sheet1" and assign column headings. */

DDE INITIATE Sheet1 FRAME DDEframe APPLICATION "Excel" TOPIC "Sheet1".
DDE SEND Sheet1 SOURCE "Name"      ITEM "R1C1".
DDE SEND Sheet1 SOURCE "YTD Sales" ITEM "R1C2".
DDE SEND Sheet1 SOURCE "State"     ITEM "R1C3".
                         .
                         .
                         .
DDE TERMINATE Sheet1.
```

Notes

- Before closing a DDE conversation, remove all advise links in the conversation using the DDE ADVISE statement.
- Closing this conversation makes *ddeid* unavailable for further exchanges, but any other conversations open to the same server are still available.
- For more information on using the DDE protocol to exchange data with non-OpenEdge applications, see *OpenEdge Development: Programming Interfaces*.

See also

DDE ADVISE statement, DDE INITIATE statement

DECIMAL function

Converts an expression of any data type to a decimal value.

Syntax

```
DECIMAL ( expression )
```

expression

If *expression* is a character, then it must be valid for conversion into a number. (For example, 1.67 is valid but 1.x3 is not valid.) If *expression* is logical, then the result is 0 if *expression* is FALSE and 1 if *expression* is TRUE. If *expression* is a date, then the result is the number of days from 1/1/4713 B.C. to that date. If the value of *expression* is the Unknown value (?), then the result is also the Unknown value (?).

Example

The example procedure lets the user enter new values for credit-limit in a special form. If the user enters the letter a, the procedure uses the standard a credit of 5000; if the user enters b, the procedure uses a value of 2000; if the user presses **RETURN**, the procedure uses a value of 1000. Otherwise, the user can enter any value for credit-limit. The DECIMAL function converts the value entered into a decimal value.

r-decml.p

```
DEFINE VARIABLE new-max AS CHARACTER FORMAT "x(10)".

REPEAT:
    PROMPT-FOR customer.cust-num WITH FRAME credit.
    FIND customer USING cust-num.
    DISPLAY cust-num name credit-limit WITH FRAME credit DOWN.
    DISPLAY "Enter one of:" SKIP(1)
            "a = 5000" SKIP
            "b = 2000" SKIP
            "RETURN = 1000"
            "A dollar value"
            WITH FRAME vals COLUMN 60.
    SET new-max WITH FRAME credit.
    IF new-max = "a" THEN credit-limit = 5000.
    ELSE IF new-max = "b" THEN credit-limit = 2000.
    ELSE IF new-max > "0" AND new-max < "999,999.99" THEN
         credit-limit = DECIMAL(new-max).
    ELSE credit-limit = 1000.
    DISPLAY credit-limit WITH FRAME credit.
END.
```

See also

INTEGER function, STRING function

DECRYPT function

Converts encrypted data (a binary byte stream) to its original source format, and returns a MEMPTR containing the decrypted data.

Note: You must use the same cryptographic algorithm, initialization vector, and encryption key values to encrypt and decrypt the same data instance.

Syntax

```
DECRYPT ( data-to-decrypt [ , encrypt-key [ , iv-value [ , algorithm ] ] ] )
```

data-to-decrypt

The encrypted data to decrypt. The value may be of type RAW or MEMPTR.

encrypt-key

An optional RAW expression that evaluates to the encryption key (a binary value) originally used to encrypt the specified data. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-KEY attribute is used. If the value of the SYMMETRIC-ENCRYPTION-KEY attribute is also the Unknown value (?), Progress generates a runtime error.

Progress compares the size of the specified encryption key to the key size specified by the cryptographic algorithm. If the key sizes are inconsistent, Progress generates a runtime error.

You can use the GENERATE-PBE-KEY function to regenerate the same encryption key originally used to encrypt the specified data as long as you specify the same password string, hash algorithm, number of iterations, and salt value.

Note: Do not use the GENERATE-RANDOM-KEY function to generate this encryption key. The random key it generates will always be different than the key originally used to encrypt the specified data.

You are responsible for generating, storing, and transporting this value.

iv-value

An optional RAW expression that evaluates to the initialization vector value to use with the specified encryption key in the original encryption operation. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-IV attribute is used.

algorithm

An optional CHARACTER expression that evaluates to the name of the symmetric cryptographic algorithm originally used to encrypt the specified data instance. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-ALGORITHM attribute is used.

For a list the supported cryptographic algorithms, see the SYMMETRIC-SUPPORT attribute reference entry.

See also ENCRYPT function, SECURITY-POLICY system handle

DEFINE BROWSE statement

Defines and creates either a static read-only browse widget or a static updateable browse widget.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ [ NEW ] SHARED ] [ PRIVATE ] BROWSE browse-name
  QUERY query-name
    [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ] [ NO-WAIT ]
  DISPLAY
    { column-list | record [ EXCEPT field ... ] }
    [ browse-enable-phrase ]
    { browse-options-phrase }
    [ CONTEXT-HELP-ID expression ]
    [ DROP-TARGET ]
    [ TOOLTIP tooltip ]
```

`NEW SHARED BROWSE browse-name`

Defines and identifies a browse widget that can be used by other procedures. When the procedure containing this statement ends, the browse is no longer available.

`SHARED BROWSE browse-name`

Defines and identifies a browse that was created in another procedure with the DEFINE NEW SHARED BROWSE statement.

`[ PRIVATE ] BROWSE browse-name`

Defines and identifies a browse widget as a data member for a class, and optionally specifies an access mode for that data member. Do not specify the access mode when defining a browse widget for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

BROWSE *browse-name*

Identifies the name of the browse you want to define for the query. You can define the browse widget in a procedure or a method within a class.

QUERY *query-name*

The name of the query to browse. You must have previously defined or opened the query.

[SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK]

Specifies the locking mode for records retrieved by the browse widget. The default locking mode is NO-LOCK. To control locking during preselection for a query associated with a browse widget, use the SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK option in the OPEN QUERY statement that opens the query.

NO-WAIT

Specifies not to wait for a record that is currently locked by another process. Instead, the record in conflict will be made available in NO-LOCK mode and the LOCKED function for that record will return TRUE.

DISPLAY *column-list*

Specifies the column items to display in the browse. Note that the *column-list* cannot contain widgets other than fill-ins and the *column-list* cannot contain SKIP options.

```
DISPLAY
  { expression
      [ column-format-phase ]
      [ @ base-field ]
  } ...
```

expression

A field name, variable, constant, or expression to display in each iteration of the browse frame.

column-format-phrase

Specifies the format for a value displayed in the browse. The *column-format-phrase* is a subset of the Format phrase.

```
[ FORMAT expression ]
  [ LABEL label | NO-LABELS ]
  [ WIDTH n ]
  [ COLUMN-FONT expression ]
  [ COLUMN-LABEL label ]
  [ COLUMN-DCOLOR expression ]
  [ COLUMN-BGCOLOR expression ]
  [ COLUMN-FGCOLOR expression ]
  [ COLUMN-PFCOLOR expression ]
  [ LABEL-FONT constant ]
  [ LABEL-DCOLOR expression ]
  [ LABEL-BGCOLOR expression ]
  [ LABEL-FGCOLOR expression ]
```

WIDTH *n*

Specify a width for the browse column. *n* represents a multiplier of the average character width of the column font. Specifying a width smaller than the format string creates a scrolling browse cell, if the column is updateable.

For more information on FORMAT strings and label options, see the Format phrase reference entry. The column and label color and font options work like those specified in the *browse-options-phrase*. If color or fonts are specified with this phrase, they only affect the specific column and override similar options specified in the *browse-options-phrase*.

@*base-field*

The *base-field* must be the name of a field or variable; it cannot be an expression or constant.

Progress reserves enough space for the *base-field* to hold the longest format displayed there. All right-justified fields (numeric fields that do not use side labels) are right justified within the reserved area.

To determine the format to use for displaying the expression at the *base-field*, Progress looks at the following and uses the first format that applies:

- An explicit format used with the *expression*.
- If the expression is a character string constant, a format that accommodates that string.
- If the data type of the expression matches that of the *base-field*, the format of the *base-field*.
- The standard format of the expression as if it were displayed without a *base-field*.

DISPLAY *record*

Specifies the record you want to display. If you specify a record, all fields from the record are displayed unless you use the EXCEPT option to eliminate specific fields.

See the Record phrase reference entry for more information.

EXCEPT *field* . . .

Specifies fields that are not displayed in the browse. You can use the EXCEPT option only if you specify a record name in the DISPLAY option.

browse-enable-phrase

Specifies which fields in the *column-list* are enabled for input.

```
ENABLE
  { { field [ HELP string ]
      [ VALIDATE ( condition , msg-exp ) ]
      [ AUTO-RETURN ]
      [ DISABLE-AUTO-ZAP ]
    } ...
    ALL [ EXCEPT field ]
  }
```

List each field or variable from the *column-list* that you want enabled. Specify ALL to specify every item in the *column-list*. Use the EXCEPT option to exclude specific items when you use the ALL option.

For each field or variable, you can also specify custom help and validation, as shown in the next two entries.

HELP *string*

Represents a character string that you want to display whenever the user enters the frame field for the field or variable. When the user leaves the frame field, Progress removes the help string from the message area. You must enclose the string in quotation marks ("").

VALIDATE (*condition*, *msg-expression*)

Specifies an expression that you want to validate against the data entered into a the browse cell. The *condition* is a Boolean expression (a constant, field name, variable name, or expression) whose value is TRUE or FALSE.

When you use the VALIDATE option to validate a specific cell, any reference to that cell in *condition* is assumed to be the new input value. For example, in the *browse-enable phrase* below, the promise-date that is compared to the order-date is the new user input, not the existing data:

```
ENABLE promise-date VALIDATE(promise-date > order-date,
  "Promise date must be later than order date").
```

To validate a new value against another new value, use the INPUT qualifier, as shown below:

```
ENABLE order-date promise-date VALIDATE(promise-date > INPUT order-date,
"Promise date must be later than order date").
```

If the value of *condition* is FALSE, use *msg-expression* to display a specific message. You must enclose *msg-expression* in quotation marks ("").

Progress processes validation criteria whenever the user attempts to leave the browse cell. If the cell value is not valid, Progress displays *msg-expression* in the message area, causes the terminal to beep, and does not advance out of the browse cell.

If the user tabs to a cell, makes no changes, and leaves the cell, Progress does not process the validation criteria specified with the VALIDATE option until the you press **GO** (**F1**). If the user presses **ENDKEY** or **END-ERROR**, or an error occurs, Progress does not test the validation criteria specified with the VALIDATE option.

If the input source for the procedure is a table, Progress validates each input field (except those with a value of "-"). If the result of the validation is FALSE, *msg-expression* is displayed and Progress treats the validation as an error.

To suppress the Data Dictionary validation criteria for a cell, use this VALIDATE option:

```
VALIDATE(TRUE,"")
```

AUTO-RETURN

Indicates whether Progress should behave as if the user pressed the **RETURN** key.when the user enters the last allowable character in a browse cell of the specified browse-column.

DISABLE-AUTO-ZAP

Indicates whether Progress should ignore the value of the browse-column's AUTO-ZAP attribute and assume it is FALSE.

browse-options-phrase

Specifies options that affect the browse widget as a whole. The options affect both the layout and the function of the browse widget. (Note that you cannot include aggregate-phrases (TOTAL, MIN, etc.) in this phrase.) This is the syntax for *browse-options-phrase*:

```
WITH
  { [ constant ] DOWN [ WIDTH width ] | [ size-phrase ] }
  [ FGCOLOR expression ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ PFCOLOR expression ]
  [ LABEL-FONT expression ]
  [ LABEL-DCOLOR expression ]
  [ LABEL-FGCOLOR expression ]
  [ LABEL-BGCOLOR expression ]
  [ MULTIPLE | SINGLE ]
  [ SEPARATORS | NO-SEPARATORS ]
  [ NO-ASSIGN ]
  [ NO-ROW-MARKERS ]
  [ NO-LABELS ]
  [ NO-BOX ]
  [ FONT constant ]
  [ title-phrase ]
  [ NO-VALIDATE ]
  [ NO-SCROLLBAR-VERTICAL | SCROLLBAR-VERTICAL ]
  [ ROW-HEIGHT-CHARS | ROW-HEIGHT-PIXELS ] row-height
 [ FIT-LAST-COLUMN ]
 [ EXPANDABLE ]
 [ NO-EMPTY-SPACE ]
  [ DROP-TARGET ]
  [ NO-AUTO-VALIDATE ]
```

[*constant* **]** DOWN **[** WIDTH *width* **]**

The *constant* value is the number of rows displayed in the browse and must be at least 2. You can optionally specify the width of the browse, where *width* is the width of the browse in character units.

A *browse-options-phrase* must contain a DOWN option or a *size-phrase*.

size-phrase

Specifies the outer size of the browse border. When this option is used instead of the DOWN option, Progress determines the number of rows that can be displayed in the browse. Following is the syntax for *size-phrase*:

{ SIZE SIZE-PIXELS **}** *width* BY *height*

For more information on *size-phrase*, see the SIZE phrase reference entry.

A *browse-options-phrase* must contain a DOWN option (optionally with a WIDTH option) or a *size-phrase*.

FGCOLOR *expression*

Specifies the foreground color for the browse in graphical environments, but not the label foreground color. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in character environments.

BGCOLOR *expression*

Specifies the background color for the browse in graphical environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in character environments.

DCOLOR *expression*

Specifies the display color for the browse in character environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in graphical environments.

PFCOLOR *expression*

Specifies the prompt color for the browse in character environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in graphical environments.

LABEL-FONT *constant*

Specifies the font of the browse labels.

LABEL-DCOLOR *expression*

Specifies the display color for the browse labels in character environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in graphical environments.

LABEL-FGCOLOR *expression*

Specifies the foreground color for the browse labels in graphical environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in character environments.

LABEL-BGCOLOR *expression*

Specifies the background color for the browse labels in graphical environments. The value of *expression* must be an integer value that specifies an entry in the color table. This option is ignored in character environments.

[MULTIPLE | SINGLE]

Specifies whether multiple rows can be selected from the browse or only a single row at one time. The default is SINGLE.

[SEPARATORS | NO-SEPARATORS]

Specifies whether row and column separators are displayed within the browse. The default is NO-SEPARATORS.

NO-ASSIGN

Disables automatic writes on new data in an updateable browse. If this option is not specified, data entered into an updateable browse is assigned on any action that results in a ROW-LEAVE event. This option is intended for use with user-defined triggers on the ROW-LEAVE event. Essentially, when this option is specified, you must make all data assignments by way of the updateable browse.

```
ON ROW-LEAVE OF my-browse DO:
  IF Customer.State:SCREEN-VALUE IN BROWSE my-browse NE "MA" THEN DO:
    MESSAGE "Customer is out of state."
    RETURN NO-APPLY.
  END.

  /* Your code. Transaction scope is up to you. */

  FIND CURRENT customer.

  IF NOT CURRENT-CHANGED(customer) THEN
    ASSIGN INPUT BROWSE field1 field2 field3 field4.
  ELSE MESSAGE "Record has changed since last read.".

END.
```

In the above example, the code looks for a special case where automatic database writes are not desirable and prevents them. The body of the trigger handles other processing before proceeding to commit the changes. First the trigger refinds the current customer record and then uses the CURRENT-CHANGED function to see if it has changed while the user was updating the browse cells. If it has not changed, the changes are committed. If it has changed, the trigger would handle that condition, too.

Note that an ASSIGN statement with the INPUT BROWSE option can be mixed with other assignment types, as shown:

```
ASSIGN
  Name
  a = b
  INPUT FRAME my-frame c d
  INPUT BROWSE my-browse order-date promise-date
  INPUT e.
```

NO-ROW-MARKERS

By default, an updateable browse displays row markers, which allow the user to select currently displayed rows in an updateable browse widget. This option prevents row markers from being displayed.

NO-LABELS

Does not display labels above the columns of the browse.

NO-BOX

Does not display a box around the browse. If you do not use this option, Progress displays a box around the data you are displaying.

If you are sending data to a device other than a terminal and you do not use this option, Progress omits the sides and bottom line of the box and replaces the top line with blanks.

FONT *constant*

Specifies the font of the browse. The title and labels also use this font, unless otherwise specified.

title-phrase

Displays a title as part of the top line of the box around the browse. For example:

```
TITLE
  [ BGCOLOR expression ] [ DCOLOR expression ]
  [ FGCOLOR expression ] [ FONT expression ]
  [ title-string ]
```

The *title-string* is a constant, field name, variable name, or expression whose result is a character value. The *expression* is the value you want to display as a title. If *title-string* is a constant character string, it must be surrounded by quotes (""). Progress automatically centers *title-string* in the top line of the browse box.

You can use the BGCOLOR and FGCOLOR options to specify the background and foreground color of the title under a graphical user interface. You can use the DCOLOR option to specify the color of the title under a character user interface.

NO-VALIDATE

Tells Progress to ignore the validations conditions in the schema for all fields in the browse.

Since browses do not inherit the NO-VALIDATE option from a parent frame, if you want a browse to have this option, you must specify it explicitly.

NO-SCROLLBAR-VERTICAL | SCROLLBAR-VERTICAL

Indicates whether the browse displays a vertical scrollbar. The default is for the vertical scrollbar to appear.

ROW-HEIGHT-CHARS | ROW-HEIGHT-PIXELS *row-height*

An INTEGER representing the row height in characters or pixels.

This option applies to graphical interfaces only.

The ROW-HEIGHT-CHARS and ROW-HEIGHT-PIXELS options let you specify the browse's row height, in either characters or pixels.

FIT-LAST-COLUMN

Allows the browse to be displayed so that there is no empty space to the right and no horizontal scroll bar by potentially widening or shrinking the last browse column's width.

This option applies to graphical interfaces only.

When this option is specified, and the last browse column can be fully or partially displayed in the browse's viewport, then the last browse column's width is adjusted so that it fits within the viewport with no empty space to its right and no horizontal scroll bar.

If the last browse column is fully contained in the viewport with empty space to its right, it grows so that its right edge is adjacent to the vertical scroll bar.

If the last browse column extends outside the viewport, it shrinks so its right edge is adjacent to the vertical scroll bar and the horizontal scroll bar is not needed.

The default value is FALSE.

Note: The FIT-LAST-COLUMN option performs the same function as the EXPANDABLE option. Progress Software recommends that you use the FIT-LAST-COLUMN option instead of the EXPANDABLE option. This recommendation includes replacing EXPANDABLE with FIT-LAST-COLUMN in your current code.

EXPANDABLE

If you set a browse's EXPANDABLE option to TRUE, Progress extends the right-most browse-column horizontally to the browse's right edge, if necessary, to cover any white space that might appear there — unless you explicitly set the width of the right-most browse-column using the WIDTH-CHARS or WIDTH-PIXELS option. The expansion of the right-most browse-column might occur anytime the browse or another browse-column is resized.

The right-most browse-column expands only when there is no horizontal scroll bar. This is because when there is a horizontal scroll bar, no white space appears between the right edge of the right-most browse-column and the right edge of the browse.

Note: The EXPANDABLE option performs the same function as the FIT-LAST-COLUMN option. Progress Software recommends that you use the FIT-LAST-COLUMN option instead of the EXPANDABLE option. This recommendation includes replacing EXPANDABLE with FIT-LAST-COLUMN in your current code.

NO-EMPTY-SPACE

Allows the b**rowse to display** with no empty space to the right and no horizontal scroll bar.

You cannot specify both NO-EMPTY-SPACE and FIT-LAST-COLUMN for the DEFINE BROWSE statement. If you specify both, the compiler displays an error message. If you set either the NO-EMPTY-SPACE option or the DEFINE BROWSE option to TRUE and one of them is already TRUE, a warning message displays at run time.

NO-AUTO-VALIDATE

Tells Progress to compile into the code all relevant validations it finds in the OpenEdge Data Dictionary, but to run the validations only when the code for a browse or for a browse-column specifically invokes the VALIDATE() method.

CONTEXT-HELP-ID *expression*

An integer value that specifies the identifier of the help topic for this browse in a help file specified at the session, window, or dialog box level using the CONTEXT-HELP-FILE attribute.

DROP-TARGET

Indicates whether the user can drop a file onto the object.

TOOLTIP *tooltip*

Allows you to define a help text message for a browse widget. Progress automatically displays this text when the user pauses the mouse pointer over a browse widget for which a ToolTip is defined. You can add or change the TOOLTIP option at any time.

If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed from the browse. No ToolTip is the default. ToolTips are supported in Windows only.

Examples

This procedure sets up a read-only browse widget for the customer table. The browse displays the Cust-num and Name fields. A separate frame, `f2`, displays more information on the currently chosen customer.

r-browse.p

```
DEFINE QUERY q1 FOR customer.
DEFINE BROWSE b1 QUERY q1 DISPLAY cust-num name
    WITH 17 DOWN TITLE "Customer Browse".

DEFINE FRAME f1
             b1
        WITH SIDE-LABELS AT ROW 2 COLUMN 2.
DEFINE FRAME f2
        WITH 1 COLUMNS AT ROW 2 COLUMN 38.
ON VALUE-CHANGED OF b1
DO:
    DISPLAY customer EXCEPT comments WITH FRAME f2.
END.

OPEN QUERY q1 FOR EACH customer.
ENABLE b1 WITH FRAME f1.
APPLY "VALUE-CHANGED" TO BROWSE b1.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

The VALUE-CHANGED event occurs each time the user selects a row within the browse widget. The associated database record is automatically placed into the record buffer. The trigger on the VALUE-CHANGED event displays that record in frame `f2`.

The APPLY statement causes the first Customer record to display before the user selects a record.

The second example sets up an updateable browse that displays some fields from the customer table. Select a row marker to select a row. Select a cell to edit it. Select a column label to initiate a search. (The trigger on ROW-LEAVE is only necessary because the NO-ASSIGN option prevents automatic commitment of the data when the user leaves a row.)

r-brows2.p

```
DEFINE QUERY q1 FOR customer SCROLLING.
DEFINE BROWSE b1 QUERY q1 NO-LOCK DISPLAY cust-num name phone
    ENABLE name phone WITH 15 DOWN NO-ASSIGN SEPARATORS.
DEFINE VARIABLE method-return AS LOGICAL NO-UNDO.
DEFINE BUTTON button1 LABEL "New Row".
DEFINE FRAME f1
    SKIP(1)
    SPACE(8) b1 SKIP(1)
    SPACE(8) button1
        WITH NO-BOX.

ON ROW-LEAVE OF b1 IN FRAME f1 /* No-Assign Browser */
DO:
    /* If new row, create record and assign values in browse. */
    IF b1:NEW-ROW THEN DO:
        CREATE CUSTOMER.
        ASSIGN INPUT BROWSE b1 name phone.
        DISPLAY cust-num WITH BROWSE b1.
       method-return = b1:CREATE-RESULT-LIST-ENTRY().
         RETURN.
    END.
   /* If record exists and was changed in browse, update it. */
    IF BROWSE b1:CURRENT-ROW-MODIFIED then DO:
        GET CURRENT q1 EXCLUSIVE-LOCK.
        IF CURRENT-CHANGED customer THEN DO:
            MESSAGE "This record has been changed by another user."
                SKIP "Please re-enter your changes.".
            DISPLAY cust-num name phone WITH BROWSE b1.
            RETURN NO-APPLY.
        END.
        ELSE /* Record is the same, so update it with exclusive-lock */
            ASSIGN INPUT BROWSE b1 name phone.
        /* Downgrade the lock to a no-lock. */
        GET CURRENT q1 NO-LOCK.
    END.
END.

ON CHOOSE OF button1 IN FRAME f1 /* Insert */
DO:
    method-return = b1:INSERT-ROW("AFTER").
END.
OPEN QUERY q1 FOR EACH customer.
ENABLE ALL WITH FRAME f1.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW
```

Notes

- You cannot define a SHARED or NEW SHARED browse widget in a persistent procedure. If you do, Progress raises ERROR on the RUN statement that creates the procedure.
- You cannot define a SHARED or NEW SHARED browse widget in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- The vertical scrollbar is displayed with the browse by default in Windows interfaces. It may be removed by setting the SCROLLBAR-VERTICAL attribute to FALSE or by specifying NO-SCROLLBAR-VERTICAL in the DEFINE BROWSE statement. If the horizontal scrollbar is needed, it is provided by default.
- The vertical scrollbar thumb size reflects the percentage of rows that are displayed in the viewport relative to the number of rows in the results list. If all the rows have not yet been read into the results list, the Progress 4GL uses the MAX-DATA-GUESS attribute to estimate the total size.
- You must put the browse into a frame on the same procedure level on which the browse is defined. For example, you cannot define a browse in an outer procedure and then display it in a frame defined within an internal procedure.
- You cannot display a browse widget in a down frame. Progress automatically converts any frame containing a browse to a 1 down frame.
- You can modify the field values displayed in the current iteration of a browse by using the WITH BROWSE option of the DISPLAY statement. For example:

```
DISPLAY { field | { value @ field } } ...
  WITH BROWSE browse
```

- The browse widget has built-in support for the **HOME**, **END**, **PAGE-UP**, and **PAGE-DOWN** key functions.
- You can specify an application-defined widget ID for a static browse widget using the *form-item* phrase in either the FORM statement or the DEFINE FRAME statement. See the FORM statement and DEFINE FRAME statement reference entries for more information.

- Progress treats the query associated with a browse as a scrolling query. You do not have to specify SCROLLING in the DEFINE QUERY statement.
- When you execute an OPEN QUERY or REPOSITION statement for the query associated with the browse, the browse is automatically adjusted to remain in sync with the query. However, when you execute a GET statement, the browse is not adjusted. You can use the GET statement to perform background processing without affecting the browse, but you must execute a REPOSITION statement to put the query and browse back in sync.
- The record locking behavior specified for a query in the DEFINE BROWSE statement overrides the record locking behavior specified with the OPEN QUERY statement. The default record locking behavior of a browse widget is NO-LOCK. The default record locking behavior of a query defined with the OPEN QUERY statement is SHARE-LOCK. If you define a query and a browse widget for the query without explicitly defining record locking behavior, the query will have the NO-LOCK behavior.
- For an updateable browse, Progress re-gets the record with a SHARE-LOCK when the user first edits a row, if it initially has a NO-LOCK. The user then can make changes to the updateable cells in the row. When the user leaves a row with changes (moves to a new row or another widget), Progress starts a transaction and gets the record with EXCLUSIVE-LOCK and NO-WAIT. If Progress gets the record, the record is updated, the record is disconnected (removes the lock), the transaction ends, and the lock is downgraded to its original status. If the get record with EXCLUSIVE-LOCK fails, the transaction is backed out, an error message is displayed, and focus remains with the edited browse row with the changed data. To redisplay the original data, use the DISPLAY...WITH BROWSE statement.
- All LEAVE triggers for a browse row execute before the row changes are committed. If the LEAVE trigger returns a NO-APPLY, the changes are not committed.
- It is also possible to use an updateable browse to add new records and delete old ones. For a complete discussion of these techniques, see the browse chapter in *OpenEdge Development: Progress 4GL Handbook*.
- Browse widgets in Windows have user search capabilities. Special events allow you to extend these capabilities. For a complete discussion of these techniques, see the chapter on database access in *OpenEdge Development: Progress 4GL Handbook*.
- When an updateable browse enters edit mode, all selected records are deselected. Essentially, a browse in edit mode ignores multiple selections.

- The ADD-CALC-COLUMN, ADD-COLUMNS-FROM and ADD-LIKE-COLUMN methods may be used on a static browse to add a dynamic browse column. If a dynamic browse column in a static browse is made updateable, the browse is changed to a NO-ASSIGN browse and you become responsible for any database update associated with it.
- The browse's QUERY attribute can now be set to the Unknown value (?). If this is done, all browse-columns are removed.
- The browse's QUERY attribute can now be changed to any query. Previously, the new query had to have the same underlying fields as the original query. If the new query is different, all browse columns are removed. You must specify new columns with the ADD-CALC-COLUMN, ADD-COLUMNS-FROM, and ADD-LIKE-COLUMN methods.

See also ADD-CALC-COLUMN( ) method, ADD-COLUMNS-FROM( ) method, ADD-LIKE-COLUMN( ) method, CLOSE QUERY statement, CREATE BROWSE statement, CURRENT-CHANGED function, CURRENT-RESULT-ROW function, DEFINE QUERY statement, DISPLAY statement, FIND statement, FORM statement, Format phrase, Frame phrase, GET statement, NUM-RESULTS function, OPEN QUERY statement, RUN statement

DEFINE BUFFER statement

Progress provides you with one buffer for each table that you use in a procedure. Progress uses that buffer to store one record at a time from the table as the records are needed during the procedure. If you need more than one record at a time from a table, you can use the DEFINE BUFFER statement to define additional buffers for that table. If you need to share buffers among procedures, use the DEFINE SHARED BUFFER statement.

Syntax

```
DEFINE [ [ NEW ] SHARED ]
       [ PRIVATE | PROTECTED ] BUFFER buffer-name
       FOR [ TEMP-TABLE ] table-name
       [ PRESELECT ] [ LABEL label-name ]
       [ NAMESPACE-URI namespace ] [ NAMESPACE-PREFIX prefix ]
```

`NEW SHARED BUFFER` *buffer-name*

Defines and identifies a buffer that can be used by other procedures. When the procedure using this statement ends, the buffer is no longer available.

`SHARED BUFFER` *buffer-name*

Defines and identifies a buffer that was created in another procedure with the DEFINE NEW SHARED BUFFER statement.

`[ PRIVATE | PROTECTED ] BUFFER` *buffer-name*

Defines and identifies a buffer as a data member for a class, and optionally specifies an access mode for that data member. Do not specify an access mode when defining a buffer for a method within a class.

PRIVATE data members can be accessed only by the defining class. PROTECTED data members can be accessed by the defining class and any of its inheriting classes. The default access mode is PRIVATE.

Note: These options are applicable only when defining a data member for a class in a class definition (`.cls`) file.

BUFFER *buffer-name*

Identifies the name of the buffer you want to define to hold records from *table-name*. You can define the buffer in a procedure or a method within a class.

FOR [TEMP-TABLE] *table-name*

Identifies the name of the table for which you are defining an additional buffer. This can also be the built-in buffer name, `proc-text-buffer`, to define a buffer that returns table rows from a stored procedure.

To define a buffer for a table defined for multiple databases, you might have to qualify the table name with the database name. See the Record phrase reference entry for more information.

Use the TEMP-TABLE option to define a buffer for a temporary table when the temporary table has the same name as a database table. Otherwise, Progress associates the buffer with the database table by default.

PRESELECT

If you use the PRESELECT option with a DO or REPEAT block, Progress creates an internal list of the records selected. The PRESELECT option tells Progress to apply that internal list to the buffer you define. You can also use the PRESELECT option in the DEFINE SHARED BUFFER statement.

LABEL *label-name*

Specifies a label for the buffer. This label is used in error messages in place of the buffer name.

NAMESPACE-URI *namespace*

An optional CHARACTER constant that specifies the URI for the namespace of the buffer object.

NAMESPACE-PREFIX *prefix*

An optional CHARACTER constant that specifies the namespace prefix associated with the NAMESPACE-URI.

Examples

This procedure allows the user to create a new customer record. Initially, the City, State, and Country fields are not shown. After the user enters a Postal-Code value, the procedure searches for an existing customer with the same postal code. If such a customer is found, the City, State, and Country values from that record are displayed in the fields for the new record. The user can then update those fields.

r-defb.p

```
DEFINE BUFFER other-cust FOR Customer.

FORM
   Customer
   WITH FRAME cre-cust.

ON LEAVE OF Customer.Postal-Code
   DO:
      FIND FIRST other-cust WHERE other-cust.Postal-Code =
                                  Customer.Postal-Code:SCREEN-VALUE AND
                                  other-cust.Cust-num <>
                                  Customer.Cust-num NO-ERROR.
      IF AVAILABLE(other-cust)
      THEN DISPLAY other-cust.City @ Customer.City
                   other-cust.State @ Customer.State
                   other-cust.Country @ Customer.Country
                   WITH FRAME cre-cust.
      ENABLE Customer.City Customer.State Customer.Country
         WITH FRAME cre-cust.
   END.

CREATE Customer.
UPDATE Customer EXCEPT Customer.City Customer.State Customer.Country
   WITH FRAME cre-cust.
```

The following gather a group of records so that the user can enter any table name and any set of record selection criteria and then look at the records in the table that meet those criteria.

r-defb2.p

```
DEFINE VARIABLE fname AS CHARACTER
  FORMAT "x(12)" LABEL "Table name".
DEFINE VARIABLE conditions AS CHARACTER
  FORMAT "x(60)" LABEL "Conditions".

REPEAT:
  /* Get the name of a table and, optionally,
    some record selection criteria */
  UPDATE fname COLON 12 conditions COLON 12
    WITH SIDE-LABELS 1 DOWN.
  HIDE ALL.
  IF conditions <> ""
    /* Pass the table name and the record selection
       criteria as parameters */
  THEN RUN r-defb3.p fname "WHERE" conditions.
  ELSE RUN r-defb3.p fname.
END.
```

The `r-defb2.p` procedure gets the name of a table (such as customer) and a condition (such as credit-limit > 4000) and passes them as arguments to the `r-defb3.p` procedure.

r-defb3.p

```
DEFINE NEW SHARED BUFFER rec FOR {1} PRESELECT.
DEFINE VARIABLE flist AS CHARACTER EXTENT 12.
DEFINE VARIABLE I AS INTEGER.
/* Look in _File the table named in the filename variable */
FIND _File "{1}".
/* Store the table's field names in the first array */
FOR EACH _Field OF _File USE-INDEX _Field-posit:
  IF i >= 12 THEN LEAVE.
  i = i + 1.
  flist[i] = _Field._Field-name.
END.
/* Preselect records */
DO PRESELECT EACH rec {2} {3} {4} {5} {6} {7}
        {8} {9} {10} {11} {12}:
/* Pass the filenames and all field names to r-defb4.p */
RUN r-defb4.p "{1}" flist[1] flist[2] flist[3]
       flist[4] flist[5] flist[6]
       flist[7] flist[8] flist[9]
       flist[10] flist[11] flist[12].
END.
```

The r-defb3.p procedure:

- Lets you view the OpenEdge Data Dictionary. The _File table contains a record for each of your database tables.
- Lets you look up a record for a customer table. For example, the user supplies Customer as a table name; the FIND statement in the r-defb3.p procedure translates to FIND _File Customer. The FIND statement finds, in _File, the record for the customer table.
- Lets you view the _Field table in the OpenEdge Data Dictionary. The _Field table contains a single record for each of your database fields. The FOR EACH statement reads the name of each of those fields into the first array variable. If the table name is Customer, the first array variable contains the names of each of the fields in the Customer table.
- Lets you select records. For example, the user supplies the condition credit-limit > 4000 in the table name. The DO PRESELECT EACH rec statement translates to DO PRESELECT EACH rec WHERE max-credit > 4000. Progress goes through the customer table and selects the records that meet the criteria. It creates a temporary table containing a pointer to each of those records. This list of preselected records is associated with the rec buffer.
- Runs r-defb4.p, passing the table name (Customer) and the names of all of the fields in that table.

The r-defb4.p procedure has access to the rec buffer (and through it to the set of preselected records). This connection is made by using PRESELECT on the DEFINE SHARED BUFFER statement. The r-defb4.p procedure displays those records.

r-defb4.p

```
DEFINE SHARED BUFFER rec FOR {1} PRESELECT.

REPEAT:
  FIND NEXT rec.
  display {2} {3} {4} {5} {6} {7} {8} {9} {10}
  {11} {12} {13} WITH 1 COLUMN 1 DOWN.
END.
```

Because r-defb3.p and r-defb4.p use run-time argument passing, they cannot be precompiled. Having separate versions of r-defb4.p for each table and running the appropriate one in r-defb3.p, should improve response time. This approach is worthwhile if there are many lines of code in r-defb4.p a procedure.

If you define a NEW SHARED BUFFER in a procedure, then call a subprocedure that puts a record into that buffer, and display the buffer in the main procedure, Progress displays this message:

```
Missing FOR, FIND or CREATE for customer.
```

This message is displayed when the FIND statement is not in the main procedure:

```
/* Main procedure */

DEFINE NEW SHARED BUFFER x FOR customer.
RUN proc2.p.
DISPLAY x.
```

```
/* proc2.p */

DEFINE SHARED BUFFER x FOR customer.
FIND FIRST x.
```

To avoid this, explicitly scope the customer record to the main procedure block. For example:

```
/* Main procedure */

DEFINE NEW SHARED BUFFER x FOR customer.
RUN proc2.p.
DO FOR x:
  DISPLAY x.
END.
```

Notes

- You cannot define a SHARED or NEW SHARED buffer in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- Every statement that uses a table name to refer to the default buffer can also use the name of a defined alternate buffer.
- All data definitions and field names are associated with a table, not a buffer. Data definitions and field names remain the same no matter what buffer you use.

- If two buffers contain the same record, a change to one of the buffers is automatically reflected in the other buffer.
- You can pass a buffer as a parameter to a procedure.
- A SHARED buffer remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the buffer as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED buffer, Progress includes the persistent procedure in the resolution of the corresponding NEW SHARED buffer as though the procedure were on the procedure call stack.
- If you define a temporary table with the same name as a database table and then you define a buffer for that table name, the buffer will be associated with the database table, not with the temporary table, by default. Use the TEMP-TABLE option to define a buffer for a temporary table when the temporary table has the same name as a database table.
- For more information on using the built-in buffer name `proc-text-buffer`, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

See also DEFINE PARAMETER statement, RUN statement, RUN STORED-PROCEDURE statement

DEFINE BUTTON statement

The DEFINE BUTTON statement defines a static push button for use within the current procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ PRIVATE ] BUTTON button
  [ AUTO-GO | AUTO-ENDKEY ]
  [ DEFAULT ]
  [ BGCOLOR expression ]
  [ CONTEXT-HELP-ID expression ]
  [ DCOLOR expression ]
  [ DROP-TARGET ]
  [ FGCOLOR expression ]
  [ FONT number ]
  [ IMAGE-DOWN image-phrase ]
  [ { IMAGE | IMAGE-UP } image-phrase ]
  [ IMAGE-INSENSITIVE image-phrase ]
  [ MOUSE-POINTER name ]
  [ LABEL label ]
  [ LIKE button ]
  [ PFCOLOR expression ]
  [ size-phrase ]
  [ NO-FOCUS [ FLAT-BUTTON ] ]
  [ NO-CONVERT-3D-COLORS ]
  [ TOOLTIP tooltip ]
  { [ trigger-phrase ] }
```

[PRIVATE] BUTTON *button*

Defines and identifies a button widget as a data member for a class, and optionally specifies an access mode for that data member. Do not specify the access mode when defining a button widget for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

BUTTON *button*

Specifies the name of the button. You can define the button widget in a procedure or a method within a class.

AUTO-END-KEY

Specifies that when you choose this button, Progress applies the ENDKEY event to the frame.

AUTO-GO

Specifies that when you choose this button, Progress applies the GO event to the frame.

DEFAULT

Specify DEFAULT to indicate that the button is a default button. A default button is one that handles all RETURN events when no other RETURN-enabling widget in the frame or dialog box has focus. RETURN-enabling widgets include any field-level widget for which a RETURN trigger is defined, or any button, whether or not it has a trigger defined. Thus, if a button has focus, that button handles the next RETURN event. If any other field-level widget without a RETURN trigger has focus, the default button handles the next RETURN event.

To make the button the default button for the frame in which it resides, you must also set the frame's DEFAULT-BUTTON option.

BGCOLOR *expression*

Supported only for backward compatibility.

Specifies the background color for the button in a graphical user interface.

CONTEXT-HELP-ID *expression*

An integer value that specifies the identifier of the help topic for this button in a help file specified at the session, window or dialog box level using the CONTEXT-HELP-FILE attribute.

DCOLOR *expression*

Specifies the display color for the button in character interfaces. This option is ignored in graphical interfaces.

FGCOLOR *expression*

Supported only for backward compatibility.

Specifies the foreground color for the button in a graphical user interface.

FONT *number*

Specifies the font for the button label. The value *number* must be an expression that resolves to an integer value. That integer must be associated with a specific font in your system environment files.

{IMAGE | IMAGE-UP } *image-phrase*

An image that you want to appear within the button when the button is in its up state. If the image does not have a down state, for code readability you might want to use the IMAGE option instead of the IMAGE-UP option.

The IMAGE | IMAGE-UP *image-phrase* option is ignored in character interfaces.

The syntax of *image-phrase* is as follows:

```
FILE name
  [ { IMAGE-SIZE | IMAGE-SIZE-CHARS | IMAGE-SIZE-PIXELS }
      width BY height
  ]
  [ FROM { { X n Y n } | { ROW n COLUMN n } } ]
```

For more information on this syntax, see the Image phrase reference entry.

IMAGE-DOWN *image-phrase*

An image that you want to appear within the button when the button is in its down state. The IMAGE-DOWN option is ignored in character interfaces.

For more information, see the Image phrase reference entry.

Note: The Progress 4GL draws the 3D effect only if a button has an up image, but no down image.

IMAGE-INSENSITIVE *image-phrase*

An image you want to appear within the button when the button is in its insensitive (disabled) state. This option is ignored in character interfaces.

For more information, see the Image phrase reference entry.

MOUSE-POINTER *name*

Specifies the mouse pointer for the button. The character value *name* is either the name of a Progress predefined pointer, or the name of a Windows `.cur` file that defines a pointer or an `.ani` file that contains an animated cursor.

LABEL *label*

The label displayed on the button. The name should describe the action invoked when the button is chosen. The value of *label* must be a string enclosed in quotes. The default label is the button name. If you use the LIKE *button* option and you do not use the LABEL option, the button inherits the label of the button you name.

You can indicate a character within the label to be used as a navigation mnemonic in Windows. Indicate the character by preceding it with an ampersand (&). When the button is displayed, the mnemonic is underlined. The user can choose the button by pressing **ALT** and the underlined letter. If you specify more than one button with the same mnemonic, Progress transfers focus to each of these in tab order when you make a selection.

To include a literal ampersand within a label, specify a double ampersand (&&).

LIKE *button*

Indicates the name of a defined button whose characteristics you want to use for a new button. If you name a button with this option, you must have defined that button earlier in the procedure. You can override the label, image, and on phrase by using the LABEL, IMAGE, and *on-phrase* options. If you do not use these options, the button takes on the characteristics of the button you name.

PFCOLOR *expression*

Specifies the prompt-for color for the button in character interfaces. This option is ignored in graphical interfaces.

size-phrase

Specifies the outside dimensions of the button widget. Following is the syntax for size-phrase:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

If you specify SIZE or SIZE-CHARS, the units are characters; if you specify SIZE-PIXELS, the units are pixels. For character units, the values *width* and *height* must be decimal constants; for pixel units, they must be integer constants. For more information, see the SIZE phrase reference entry.

If no size is specified, Progress calculates a default size for the button. This calculation adds the button's border thickness (that is, the combination of 3D shadows and highlights, and the focus rectangle) to the up image size defined by the IMAGE | IMAGE-UP *image-phrase* option. However, the thickness of the border depends on whether the button has dual images (up and down images) and whether it is a FLAT-BUTTON or NO-FOCUS button.

Table 23 explains how many pixels the image size expands based on the button size.

Table 23: Determining button border thickness

Button image	NO-FOCUS status	FLAT-BUTTON status	Border thickness
Up image only	No	No	7 pixels (2 pixels for the focus rectangle, 5 pixels for the 3D shading)
Up and down image	No	No	4 pixels (4 pixels for the focus rectangle, 0 pixels for the 3D shading)
Up image only	Yes	No	5 pixels (0 pixels for the focus rectangle, 5 pixels for the 3D shading)
Up and down image	Yes	No	0 pixels (Progress expects the specified image to include a border).
Up image only	Yes	Yes	2 pixels
Up and down image	Yes	Yes	2 pixels

NO-FOCUS [FLAT-BUTTON]

Specifies that the button should not accept focus. A button for which the NO-FOCUS attribute is defined will not take focus when the mouse is clicked on it, and it will not accept keyboard input. Also, Progress will not generate ENTRY or LEAVE events for the button. NO-FOCUS buttons behave similarly to standard Windows toolbar buttons. The NO-FOCUS option is supported in Windows only.

A button with the NO-FOCUS attribute is not added to its parent frame's tab order. However, if the NO-FOCUS attribute is switched from TRUE to FALSE before the button is realized, the button is added to the end of its parent frame's tab order. Switching the NO-FOCUS attribute from FALSE to TRUE before realization removes the button from its parent frame's tab order.

Note: If a frame that contains a NO-FOCUS button does not itself have focus, the frame does not receive focus when the button is pushed. In this situation, frame entry or leave events are not generated. Focus stays on the current widget when a NO-FOCUS button is pushed, even across multiple frames in a window.

FLAT-BUTTON

A flat button is a new style of button which is two-dimensional until the mouse passes over it, at which time, a 3D border appears.

NO-CONVERT-3D-COLORS

Specifies that the colors of the button's images (that is, up, down, and insensitive) are not converted to the system 3D colors. By default, Progress converts shades of gray in an image to the corresponding system 3D color. Using the NO-CONVERT-3D-COLORS option overrides this default behavior. The NO-CONVERT-3D-COLORS option is supported in Windows only.

Table 24 describes the conversion process.

Table 24: 3D-color conversions for buttons

If the color is...	And the original Red-Green-Blue (RGB) color value is...	Then the new converted system color is...
White	(255, 255, 255)	System button highlight color.
Light Gray	(192, 192, 192)	System button face color.
Dark Gray	(128, 128, 128)	System button shadow color.
Black	(0, 0, 0)	System button text color.

During a session, if Windows notifies Progress that the system colors have changed, the button's images are re-loaded and converted to the new system colors, unless the NO-CONVERT-3D-COLORS option is specified.

`TOOLTIP` *`tooltip`*

Allows you to define a help text message for a button. Progress automatically displays this text when the user pauses the mouse pointer over the button.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed from the button. No ToolTip is the default. ToolTips are supported in Windows only.

`DROP-TARGET`

Indicates whether you want to be able to drop a file onto the object.

`trigger-phrase`

Specifies application triggers for the button.

For more information, see the Trigger phrase reference entry.

Example

This procedure defines two buttons, positions the buttons within a form, assigns triggers to the buttons with ON statements, and enables the buttons by referencing them in an ENABLE statement:

r-button.p

```
DEFINE BUTTON more-button LABEL "More".
DEFINE BUTTON next-button LABEL "Next".

FORM more-button next-button
   WITH FRAME but-frame ROW 1.

FORM Customer.cust-num name
   WITH FRAME brief ROW 4.

FORM customer EXCEPT cust-num name
   WITH FRAME full ROW 7.

ON CHOOSE OF more-button
   DISPLAY customer EXCEPT cust-num name WITH FRAME full.

ON CHOOSE OF next-button
   DO:
      HIDE FRAME full.
      FIND NEXT customer NO-ERROR.
      DISPLAY cust-num name WITH FRAME brief.
   END.

FIND FIRST customer.
DISPLAY cust-num name WITH FRAME brief.

ENABLE more-button next-button WITH FRAME but-frame.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

When the procedure is run, the first customer's number and name are initially displayed. The user can choose either the MORE button to see the entire customer record or the NEXT button to see the next customer's number and name.

The following example sets up a browse that allows you to drop a file on the browse:

```
DEFINE BUTTON button-1 LABEL ''Drop Here'' DROP-TARGET.
```

Notes

- When a frame receives a default RETURN event, it actually sends a CHOOSE event to the default button.
- To create the static button you are defining, you must define a static frame that contains the button. Each frame you define that contains the same button creates an additional instance of that button. The widget handle for a static button is not available until the button is created.
- You must enable a button to make it available to the user. You can enable a button by setting its SENSITIVE attribute or by referencing it in an ENABLE or UPDATE statement.
- On a character-based terminal, a button appears as the label enclosed in angle brackets (< >). The user can move the mouse pointer to the button by pressing **TAB** or arrow keys. The user can then choose the button by pressing **SPACEBAR** or **RETURN**.
- You can specify an application-defined widget ID for a static button widget using the *form-item* phrase in either the FORM statement or the DEFINE FRAME statement. See the FORM statement and DEFINE FRAME statement reference entries for more information.
- To make an application portable between graphical and character environments, you can specify an image and a label for a button. In graphical environments, the image is used and the label is ignored; in character environments, the label is used and the image is ignored.
- If you specify a size for a button, the button is not affected by changes to the size of any contained image. If you do not specify a size for the button, the button changes size to fit the image.
- In Windows, Progress supplies the following prepackaged images for the up, down, left, and right arrows: `btn-up-arrow`, `btn-down-arrow`, `btn-left-arrow`, and `btn-right-arrow`. Specify one of these items in place of a filename.

  Use these values for the IMAGE-UP option. Doing so makes the prepackaged image available to Progress in its up, down, and insensitive state, without specifying the IMAGE-DOWN and the IMAGE-INSENSITIVE options. You will also get appropriately sized arrows based on your screen resolution.
- You can apply entry to a NO-FOCUS button programmatically. Progress does not report an error. However, the button will not respond to keyboard activity.

- The Progress 4GL draws the 3D effect only if a button has an up image, but no down image. If the button has both an up image and a down image, Progress does not draw the 3D effect; the images, themselves, should be drawn with a 3D effect.
- Progress only performs the color conversion process on bitmaps (`.bmp` files) that contain 256 or fewer colors. However, you might consider using 16-color bitmaps because only the first sixteen entries in the bitmap's color table will be converted.
- Icon colors (`.ico` files) are not converted, even if CONVERT-3D-COLORS is TRUE. To ensure that an icon will be displayed properly on a button, draw icons with a transparent background.

See also FORM statement, Image phrase

DEFINE DATASET statement

Defines a static ProDataSet object that is created at compile time, and lets you name the object, identify the temp-table buffers it incorporates, and define the data relations between those buffers.

Syntax

```
DEFINE [ NEW [ SHARED ] ]
       [ PRIVATE | PROTECTED ] DATASET dataset-name
       [ NAMESPACE-URI namespace ] [ NAMESPACE-PREFIX prefix ]
       [ REFERENCE-ONLY ]
       FOR buffer-name [ , buffer-name ] ...
       DATA-RELATION [ data-rel-name ] FOR data-rel-spec
       [ DATA-RELATION [ data-rel-name ] data-rel-spec ] ...
```

`NEW SHARED DATASET` *dataset-name*

Defines and identifies a ProDataSet object that can be shared by one or more procedures called directly or indirectly by the current procedure. The ProDataSet object remains available to other procedures until the procedure that defined it ends. The called procedures must define the same ProDataSet object name as SHARED. For shared ProDataSet objects, each *buffer-name* must be the name of a shared buffer.

`SHARED DATASET` *dataset-name*

Defines and identifies a ProDataSet object that was initially defined by another procedure as NEW SHARED. For shared ProDataSet objects, each *buffer-name* must be the name of a shared buffer.

The procedure that initially defines the object determines the name. The procedures that share the object must define the object with the same name.

[PRIVATE | PROTECTED] DATASET *dataset-name*

Defines and identifies a ProDataSet object as a data member for a class, and optionally specifies an access mode for that data member. You must define the temp-tables and buffers of the ProDataSet object with the same access mode as the ProDataSet object. Do not specify an access mode when defining a ProDataSet object for a method within a class.

PRIVATE data members can be accessed only by the defining class. PROTECTED data members can be accessed by the defining class and any of its inheriting classes. The default access mode is PRIVATE.

Note: These options are applicable only when defining a data member for a class in a class definition (`.cls`) file.

DATASET *dataset-name*

Identifies the name of the ProDataSet object. You can define the ProDataSet object in a procedure, a method within a class, or an interface.

NAMESPACE-URI *namespace*

An optional CHARACTER constant that specifies the URI for the namespace of the ProDataSet object.

NAMESPACE-PREFIX *prefix*

An optional CHARACTER constant that specifies the namespace prefix associated with the NAMESPACE-URI.

REFERENCE-ONLY

Specifies that the procedure defining this ProDataSet object is using the object definition only as a reference to a ProDataSet object that is defined and instantiated in another procedure or class, and specified as a parameter in the invocation of a RUN statement, a method in a class, or a user-defined function, using either the BY-REFERENCE or BIND option. Progress does not instantiate the reference-only object.

Passing a reference-only ProDataSet object parameter to a local routine, using either the BY-REFERENCE or BIND option allows the calling routine and the called routine to access the same object instance (instead of deep-copying the parameter).

Note: If you pass the parameter to a remote procedure, Progress deep-copies the parameter on OUTPUT and the reference-only parameter is bound to that copy.

When you pass a ProDataSet parameter to a local routine using the BY-REFERENCE option, both the calling and called routines access the calling routine's object instance (and ignore the called routine's object instance). Since the called routine's object instance is ignored, you should define the object as a reference-only object. When you define a reference-only ProDataSet object in the called routine and receive it from the calling routine using the BY-REFERENCE option, Progress binds the definition of the object in the called routine to the object instance in the calling routine for the duration of the called routine. You cannot define a reference-only ProDataSet object in the calling routine and pass it to the called routine using the BY-REFERENCE option.

When you pass a ProDataSet parameter to a local routine using the BIND option, you can define a reference-only ProDataSet object in either the calling routine or the called routine as follows:

- When you define a reference-only ProDataSet object in the calling routine and pass it to the called routine using the BIND option, Progress binds the calling routine to the object instance in the called routine. The reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates. The parameter must be an OUTPUT parameter.

  Note: If you also define the ProDataSet object instance in the called routine as a reference-only object, you must bind the object in the called routine before returning to the calling routine.

- When you define a reference-only ProDataSet object in the called routine and receive it from the calling routine using the BIND option, Progress binds the called routine to the object instance in the calling routine. The reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates. The parameter must be an INPUT or INPUT-OUTPUT parameter.

In either case, you must specify the BIND option for the parameter in both the invocation of a RUN statement, a method in a class, or a user-defined function, and in the DEFINE PARAMETER statement.

Caution: Do not delete the object or routine to which a reference-only ProDataSet object is bound, or you might be left with references to an object that no longer exists.

When you define a ProDataSet object as reference-only, you must also define all member temp-tables as such. A temp-table object defined as reference-only can be a member of a reference-only ProDataSet object or a standard ProDataSet object. However, if you define a reference-only temp-table in a standard ProDataSet object, you cannot use the ProDataSet object until you bind the reference-only temp-table.

FOR *buffer-name* [, *buffer-name*] ...

Specifies a static buffer for a previously defined temp-table whose scope includes the procedure in which the ProDataSet object was defined.

DATA-RELATION [*data-rel-name*] FOR *data-rel-spec*

Specifies a data-relation object.

The `data-rel-name` argument lets you name the data-relation object. You can use this name to obtain the object's handle at runtime. This argument is optional. The default name is Relation*n* (where *n* starts at 1 for each ProDataSet object).

The `data-rel-spec` argument specifies a pair of parent and child buffers for the data-relation object using the following syntax:

```
parent-buffer-name, child-buffer-name [ field-mapping-phrase ]
[ REPOSITION ] [ NESTED ]
```

parent-buffer-name

The parent buffer in the data-relation object.

child-buffer-name

The child buffer in the data-relation object.

field-mapping-phrase

Specifies the fields in the relation using the following syntax:

```
RELATION-FIELDS (parent-field1, child-field1
        [, parent-fieldn, child-fieldn ] ...)
```

The first field in the pair is from the parent buffer, the second field is from the child buffer. When filling the ProDataSet object, Progress retrieves data for the child buffer based on an equality match between all pairs of fields unless the Data-Relation is deactivated or there is an explicit query definition for the data source of the child buffer.

You can define a query for the data source of the child buffer, or supply custom logic in response to FILL events that take over complete responsibility for filling one level of the ProDataSet object. In these cases, the *field-mapping-phrase* is not used.

When navigating the ProDataSet object, Progress filters the data in the child buffer to include only children of the current parent.

REPOSITION

Specifies the relation mode as REPOSITION. The relation mode is SELECTION, by default.

When the relation mode is SELECTION, the ProDataSet object fills the child temp-table of the relation with all records related to the current parent. When the relation mode is REPOSITION, the relation is effectively ignored during a FILL, and the child of the relation is treated as if it were a top-level buffer.

When navigating a filled ProDataSet object with a SELECTION relation, related data is filtered as it is browsed. This means the child query of the relation is filtered to make available only children of the current parent, and the query is re-opened each time the parent table is repositioned. When navigating a filled ProDataSet object with a REPOSITION relation, the child table query is always set to match all the rows in the child table, and is not re-opened when the parent changes. Only the buffer for the child is repositioned to the matching child for the current parent.

NESTED

Specifies that child rows of a ProDataSet buffer are nested within their parent rows when writing the XML representation of data or schema. This also causes the XML Schema definitions for the related temp-tables to be nested.

Notes

- You cannot define a SHARED or NEW SHARED ProDataSet object in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- You cannot define a ProDataSet object within an internal procedure, a method in a class, or a user-define function.

See also

Buffer object handle, CREATE DATASET statement, Data-relation object handle, DEFINE TEMP-TABLE statement, GET-TOP-BUFFER( ) method, NUM-REFERENCES attribute, ProDataSet object handle

DEFINE DATA-SOURCE statement

Defines a static data-source object that is created at compile time.

Syntax

```
DEFINE DATA-SOURCE [ PRIVATE | PROTECTED ] data-source-name
    FOR [ QUERY query-name ]
   [ source-buffer-phrase [ , source-buffer-phrase ] ... ]
```

[PRIVATE | PROTECTED] DATA-SOURCE *data-source-name*

Defines and identifies a data-source object as a data member for a class, and optionally specifies an access mode for that data member. Do not specify an access mode when defining a data-source object for a method within a class.

PRIVATE data members can be accessed only by the defining class. PROTECTED data members can be accessed by the defining class and any of its inheriting classes. The default access mode is PRIVATE.

Note: These options are applicable only when defining a data member for a class in a class definition (`.cls`) file.

DATA-SOURCE *data-source-name* FOR

Identifies the name of the data-source object for the specified query or database buffers. You can define the data-source object in a procedure or a method within a class.

[QUERY *query-name*]

Specifies a query that defines the buffers for the data-source object. Use this option to override the default query for the data-source object.

You can also use the QUERY attribute to override the default query, or the FILL-WHERE-STRING attribute to override the WHERE clause in the query.

source-buffer-phrase **[** *source-buffer-phrase* **]** **...**

Specifies one or more database buffers for the data-source object using the following syntax:

```
buffer-name [ KEYS ( { field1 [ , fieldn ]... | ROWID } ) ]
```

buffer-name

The name of the database buffer.

KEYS (**{** *field1* **[** , *fieldn* **]... |** ROWID **}**)

Specifies one or more database table fields that constitute a unique key that can be used by Progress to find a record in the table given a record in the temp-table that uses it as a data source.

The ROWID keyword can occur exactly once in place of a field list to use the ROWID as the key.

Note: Using ROWID as the key is not currently supported.

See also CREATE DATA-SOURCE statement, Data-source object handle, FILL( ) method, FILL-WHERE-STRING attribute, QUERY attribute

DEFINE FRAME statement

Defines and creates a frame or dialog box that can be used within a procedure or shared among several procedures.

Syntax

```
DEFINE [ [ NEW ] SHARED ] [ PRIVATE ] FRAME frame
  [ form-item ... ]
  [{ HEADER | BACKGROUND } head-item ... ]
  { [ frame-phrase ] }
```

```
DEFINE [ [ NEW ] SHARED ] FRAME frame
  record [ EXCEPT field ... ]
  { [ frame-phrase ] }
```

`NEW SHARED FRAME` *frame*

Defines and identifies a frame to be shared by a procedure called directly or indirectly by the current procedure. The called procedure must name the same frame in a DEFINE SHARED FRAME statement.

`SHARED FRAME` *frame*

Defines and identifies a frame that was created by another procedure that used the DEFINE NEW SHARED FRAME statement. When you use the DEFINE SHARED FRAME statement, you cannot name any fields or variables in that frame that are not already named in the frame described by the DEFINE NEW SHARED FRAME statement.

[`PRIVATE` **]** `FRAME` *frame*

Defines and identifies a frame widget as a data member for a class, and optionally specifies an access mode for that data member. Do not specify the access mode when defining a frame widget for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

FRAME *frame*

Identifies the name of the frame widget you are defining. You can define the frame widget in a procedure or a method within a class.

form-item

Specifies a field-level widget or value to display in the frame, or a SPACE or SKIP directive. The data specified by all form items is owned by a single field group, duplicated for each data iteration in the frame.

This is the syntax for *form-item*:

```
{     field [ format-phrase ]
   |  constant [ at-phrase | { TO n } ]
      [ BGCOLOR expression ]
      [ DCOLOR expression ]
      [ FGCOLOR expression ]
      [ FONT expression ]
      [ PFCOLOR expression ]
      [ VIEW-AS TEXT ]
      [ WIDGET-ID id-number ]
   |  SPACE [ ( n ) ]
   |  SKIP  [ ( n ) ]
}
```

field

Specifies a field-level widget to be displayed in the frame. This value cannot be an expression or a frame. To specify a child frame, you must first define the parent and child frames, and then assign the FRAME attribute of the child frame to the widget handle of the parent frame. The child frame is assigned to the same field group as other form items.

format-phrase

Specifies one or more frame attributes for a field or variable. For more information on *format-phrase*, see the Format phrase reference entry.

constant

A constant value.

at-phrase

Specifies the location of a value within the frame. The AT phrase does not left justify the data; it simply indicates the placement of the data area. This is the syntax for *at-phrase*:

```
AT { COLUMN column | COLUMN-OF reference-point }
   { ROW row | ROW-OF reference-point }
   [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
```

```
AT { X x | X-OF reference-point }
   { Y y | Y-OF reference-point }
   [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
```

```
AT n
```

For more information, see the AT phrase reference entry.

TO *n*

The number (*n*) of the column where you want the right edge of the value. The TO option does not right justify the data; it simply indicates the placement of the data area.

BGCOLOR *expression*

Specifies the background color of the form item in graphical interfaces. This option is ignored in character interfaces.

DCOLOR *expression*

Specifies the display color of the form item in character interfaces. This option is ignored in graphical interfaces.

FGCOLOR *expression*

Specifies the foreground color of the form item in graphical interfaces. This option is ignored in character interfaces.

FONT *expression*

Specifies the font of the form item.

PFCOLOR *expression*

Specifies the prompt color of the form item in character interfaces. This option is ignored in graphical interfaces.

VIEW-AS TEXT

Specifies that the form item displays as a TEXT widget rather than as a FILL-IN.

WIDGET-ID *id-number*

Specifies a widget ID for a field-level widget or value to display in a frame. The value of *id-number* must be an expression that evaluates to an even integer value between 2 and 65534, inclusive, and must be unique across all widget IDs in the window or dialog box.

If you specify an invalid ID, the compiler displays an error message. This option is supported in graphical interfaces only, and only in Windows.

SPACE (*n*)

Identifies the number (*n*) of blank spaces to insert after the expression displays. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, Progress starts a new line and discards extra spaces. If you do not use this option or you do not use *n*, Progress inserts one space between items in the frame.

SKIP (*n*)

Identifies the number (*n*) of blank lines to insert after the expression is displayed. The number of blank lines can be can be 0. If you do not use this option, Progress does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option but do not specify *n*, or if *n* is 0, Progress starts a new line unless it is already at the beginning of a new line.

record

Represents the name of the record you want to display. Naming a record is shorthand for listing each field individually, as a form item.

EXCEPT *field* . . .

Tells Progress to display all the fields in the frame except those fields listed in the EXCEPT phrase.

HEADER

Tells Progress to place the following items in a header section at the top of the frame in a separate field group from all other data. In addition to fields, variables, and constants, the frame header can contain expressions, images, and rectangles. Progress reevaluates these expressions each time it displays the frame.

When you use the HEADER option, Progress disregards OpenEdge Data Dictionary field labels for fields you name in the DEFINE FRAME statement. Use character strings to specify labels on fields you name in the frame header.

BACKGROUND

Specifies that any following frame items are displayed in the frame background, behind the data and header in a separate field group. Typically, this option is used to display images or rectangles behind the data.

head-item

A description of a value displayed in the frame header or background, or a SPACE or SKIP directive. Following is the syntax *head-item*:

```
{     expression [ format-phrase ]
   |  constant [ at-phrase | { TO n } ]
        [ BGCOLOR expression ]
        [ DCOLOR expression ]
        [ FGCOLOR expression ]
        [ FONT expression ]
        [ VIEW-AS TEXT ]
        [ WIDGET-ID id-number ]
   |  SPACE [ ( n ) ]
   |  SKIP  [ ( n ) ]
}
```

This is exactly the same as the syntax for a *form-item*, except that a *head-item* can be an expression and does not include the PFCOLOR option. If you use an expression in a

HEADER or BACKGROUND phrase, the expression is evaluated each time the frame is viewed. If you give the PAGE-TOP or PAGE-BOTTOM option for the frame, the expression is evaluated for each page. This lets you include a reference to the PAGE-NUMBER function in the frame header.

frame-phrase

Specifies additional options for the frame, including the VIEW-AS DIALOG-BOX option to define the frame as a dialog box. For more information on frame and dialog box options, see the Frame phrase reference entry.

Examples

The following example, `r-deffrm.p`, uses the DEFINE FRAME statement to set up the format of a frame. It then scopes that frame to a FOR EACH block.

r-deffrm.p

```
DEFINE VARIABLE bal-avail LIKE customer.balance
            COLUMN-LABEL "Available!Credit" NO-UNDO.

DEFINE FRAME cust-bal
  customer.cust-num
  customer.name FORMAT "X(20)"
  customer.credit-limit LABEL "Limit"
  customer.balance
  bal-avail
  WITH CENTERED ROW 3 TITLE "Available Customer Credit" USE-TEXT.

FOR EACH customer NO-LOCK WITH FRAME cust-bal:
  DISPLAY customer.cust-num
          customer.name
          customer.credit-limit
          customer.balance
          customer.credit-limit - customer.balance @ bal-avail.
END.
```

The following example defines three frames. The cust-info frame is scoped to the trigger for the b_next button where it is first referenced. Similarly, the `cust-dtl` frame is scoped to the `b_dtl` trigger. The `butt-frame` frame is scoped to the outer procedure block.

r-dffrm1.p

```
DEFINE BUTTON b_dtl LABEL "Detail".
DEFINE BUTTON b_next LABEL "Next".
DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.

DEFINE FRAME cust-info
  customer.cust-num
  customer.name FORMAT "X(20)"
  customer.phone
  WITH CENTERED ROW 4.

DEFINE FRAME cust-dtl
  customer except customer.cust-num customer.name customer.phone
  WITH CENTERED SIDE-LABELS ROW 9.

DEFINE FRAME butt-frame
  b_dtl b_next b_quit
  WITH ROW 1.

ON CHOOSE OF b_dtl
  DISPLAY customer except customer.cust-num customer.name customer.phone
    WITH FRAME cust-dtl.

ON CHOOSE OF b_next
DO:
  HIDE FRAME cust-dtl.
  FIND NEXT customer NO-LOCK NO-ERROR.
  IF NOT AVAILABLE customer
  THEN FIND LAST customer NO-LOCK.

  DISPLAY customer.cust-num
          customer.name
          customer.phone
    WITH FRAME cust-info.
END.

ENABLE ALL WITH FRAME butt-frame.

APPLY "CHOOSE" TO b_next IN FRAME butt-frame.

WAIT-FOR CHOOSE OF b_quit.
```

The following example uses a set of thin rectangles as lines to create graphic columns within a frame background:

r-bkgrnd.p

```
DEFINE VARIABLE item-tot AS DECIMAL LABEL "Value" NO-UNDO.

DEFINE RECTANGLE vline1 SIZE .4 BY 5 EDGE-PIXELS 2.
DEFINE RECTANGLE vline2 LIKE vline1.
DEFINE RECTANGLE vline3 LIKE vline1.
DEFINE RECTANGLE vline4 LIKE vline1.
DEFINE RECTANGLE vline5 LIKE vline1.
DEFINE RECTANGLE vline6 LIKE vline1.

DEFINE RECTANGLE hline SIZE 78 BY .1 EDGE-PIXELS 2.

DEFINE FRAME item-info
  item.item-num
  item.item-name
  item.on-hand
  item.re-order
  item.on-order
  item.price
  item-tot
  BACKGROUND skip(1) hline
    vline1 AT 9
    vline2 AT 25
    vline3 AT 33
    vline4 AT 42
    vline5 AT 51
    vline6 AT 65
  WITH TITLE "Inventory Current Value" CENTERED USE-TEXT 5 DOWN.

FOR EACH item NO-LOCK WITH FRAME item-info:
  DISPLAY item.item-num
    item.item-name
    item.on-hand
    item.re-order
    item.on-order
    item.price
     item.on-hand * item.price @ item-tot.
```

The following procedure defines the shared frame cust-frame. It also defines a shared variable and a shared buffer. For each customer whose customer number is less than 20, the procedure displays customer information in the cust-frame. The format for the cust-frame is defined in the `r-shrfrm.i` include file.

r-shrfrm.p

```
DEFINE NEW SHARED FRAME cust-frame.
DEFINE NEW SHARED VARIABLE csz AS CHARACTER FORMAT "x(29)".
DEFINE NEW SHARED BUFFER xcust FOR customer.

FOR EACH xcust WHERE  cust-num <=20:

      {r-shrfrm.i}  /* include file for layout of shared frame */

DISPLAY name phone address sales-rep
  city + ", " + st + " " + postal-code @ csz
  credit-limit WITH FRAME cust-frame.

RUN r-updord.p.  /* External procedure to update customer's orders */
END.
```

r-shrfrm.i

```
FORM  xcust.name         COLON 10
      xcust.phone        COLON 55
      xcust.address      COLON 1
      xcust.sales-rep    COLON 55
      csz NO-LABEL       COLON 10
      xcust.credit-limit COLON 55
      SKIP(2)
      order.order-num    COLON 10    order.order-date   COLON 30
      order.ship-date    COLON 30
      order.promise-date COLON 30  WITH SIDE-LABELS 1 DOWN CENTERED ROW 5
      TITLE   "Customer/Order Form" FRAME cust-frame.
```

After the `r-shrfrm.p` procedure displays the customer information, it calls the `r-updord.p` procedure.

The r-updord.p procedure defines the variable, frame, and buffer that were originally defined in the r-shrfrm.p procedure. However, in this second reference to the items, the keyword NEW is omitted. The r-updord.p procedure displays, and lets you update, the order information for the customer displayed in the cust-frame. The order information is displayed in the same frame.

r-updord.p

```
DEFINE SHARED FRAME cust-frame.
DEFINE SHARED VARIABLE csz  AS CHARACTER FORMAT "x(29)".
DEFINE SHARED BUFFER xcust FOR customer.

FOR EACH order OF xcust:

  {r-shrfrm.i }  /* include file for layout of shared frame */

DISPLAY order.order-num WITH FRAME cust-frame.
UPDATE order.order-date order.ship-date order.promise-date
   WITH FRAME cust-frame.
END.
```

The following example, r-fof1.p, creates a dialog box to display customer information from a query. The dialog box contains three child frames to display customer contact information (FRAME cont-fr), customer account information (FRAME acct-fr), and control buttons for moving through the query results list (FRAME ctrl-fr).

r-fof1.p *(1 of 2)*

```
DEFINE QUERY custq FOR customer.
DEFINE BUTTON bprev LABEL "<".
DEFINE BUTTON bnext LABEL ">".
DEFINE FRAME cust-fr SKIP(.5)
    SPACE(8) customer.name customer.cust-num customer.sales-rep
    customer.comments AT COLUMN 6 ROW 13.5
    WITH SIDE-LABELS TITLE "Customer Data"
        SIZE 80 BY 15 VIEW-AS DIALOG-BOX.
DEFINE FRAME cont-fr SKIP(.5)
    customer.address COLON 17 SKIP
    customer.address2 COLON 17 SKIP
    customer.city COLON 17 SKIP
    customer.state COLON 17 SKIP
    customer.postal-code COLON 17 SKIP
    customer.country COLON 17 SKIP
    customer.contact COLON 17 SKIP
    customer.phone COLON 17
    WITH SIDE-LABELS TITLE "Contact Informaion"
        SIZE 40 BY 10 AT COLUMN 1 ROW 3.
```

r-fof1.p *(2 of 2)*

```
DEFINE FRAME ctrl-fr SKIP(.12)
    SPACE(4) bprev bnext
    WITH TITLE "PREVIOUS/NEXT"
        SIZE 15 BY 2 AT COLUMN 53 ROW 10.5.
DEFINE FRAME acct-fr SKIP(.5)
    customer.balance COLON 15 SKIP
    customer.credit-limit COLON 15 SKIP
    customer.discount COLON 15 SKIP
    customer.terms COLON 15
    WITH SIDE-LABELS TITLE "Account Information"
        SIZE 38.85 BY 6 AT COLUMN 41 ROW 3.

ON CHOOSE OF bnext DO:
    GET NEXT custq.
    IF NOT AVAILABLE customer THEN GET FIRST custq.
    RUN display-proc IN THIS-PROCEDURE.
END.

ON CHOOSE OF bprev DO:
    GET PREV custq.
    IF NOT AVAILABLE customer THEN GET LAST custq.
    RUN display-proc IN THIS-PROCEDURE.
END.

FRAME cont-fr:FRAME = FRAME cust-fr:HANDLE.
FRAME acct-fr:FRAME = FRAME cust-fr:HANDLE.
FRAME ctrl-fr:FRAME = FRAME cust-fr:HANDLE.

OPEN QUERY custq PRESELECT EACH customer BY customer.name.
GET FIRST custq.
RUN display-proc IN THIS-PROCEDURE.
ENABLE ALL WITH FRAME ctrl-fr.

WAIT-FOR WINDOW-CLOSE OF FRAME cust-fr.

PROCEDURE display-proc:
    DISPLAY
        customer.name customer.cust-num customer.sales-rep
        customer.comments WITH FRAME cust-fr.
    DISPLAY
        customer.address customer.address2 customer.city customer.state
        customer.postal-code customer.country customer.contact
        customer.phone WITH FRAME cont-fr.
    DISPLAY
        customer.balance customer.credit-limit
        customer.discount customer.terms WITH FRAME acct-fr.
END.
```

Notes

- You cannot define a SHARED or NEW SHARED frame widget in a persistent procedure. If you do, Progress raises ERROR on the RUN statement that creates the procedure.
- You cannot define a SHARED or NEW SHARED frame widget in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- If you do not specify the font for a frame, Progress uses the system default font, not the font of the window. This is because Progress determines the frame layout at compile time when the window's fonts (known at runtime) are not yet available.
- You can use just one DEFINE FRAME statement per frame in a procedure.
- If you name variables or parent child frames to a shared frame, Progress does not automatically make those variables and child frames shared. If you want to share the variables and child frames among procedures, you must define each variable and frame using the SHARED option in all the sharing procedures.
- Progress scopes a newly defined frame to the block that first references the frame. (The DEFINE FRAME statement does not count as a reference.) Progress scopes a shared frame outside of the called procedure.
- The *`frame-phrase`* options specified in a DEFINE NEW SHARED FRAME statement are carried over to all corresponding DEFINE SHARED FRAME statements and cannot be overridden.
- You can use different field-level help and validation in new shared, and shared frames.
- You must define a shared frame before referencing that frame in a procedure.
- All frame fields and Frame phrase options in a shared frame must first be defined in the initial DEFINE NEW SHARED FRAME statement or an additional FORM statement in the same procedure. Procedures that share this frame only have to define fields that correspond to the fields in the initial definition plus any specified ACCUM option. Other Frame phrase options for the SHARED frames are allowed, but are ignored except for the ACCUM option. This allows you to make use of the same FORM statement in an include file for both the NEW SHARED and matching SHARED frames. See the FORM statement reference entry for more information.

- If you use an Aggregate phrase to accumulate a value within a shared frame, you must also use the ACCUM option in each procedure that uses the shared frame.
- If you define a frame to use as a DDE frame, you must realize the frame (display it) before using it as a conversation end-point. If you want the DDE frame to remain invisible during its use in a DDE conversation, set its HIDDEN attribute to TRUE after realizing the frame. For information on DDE frames, see *OpenEdge Development: Programming Interfaces*.
- If you have enabled application-defined widget IDs in your OpenEdge GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then Progress uses the value specified in the WIDGET-ID option to set the WIDGET-ID attribute for this widget when it creates the widget at runtime, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then Progress ignores this option setting at runtime.

  For more information about the WIDGET-ID attribute, see its reference entry in the "Attributes and Methods Reference" section on page 1497. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

See also DEFINE BUFFER statement, DEFINE VARIABLE statement, FORM statement, Frame phrase, RUN statement

DEFINE IMAGE statement (Windows only; Graphical interfaces only)

Defines a static image widget in a graphical interface for use in the current procedure. An image widget is a container for an operating system image file and can be displayed in a form or used as a form background.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ PRIVATE ] IMAGE image-name
  { image-phrase | LIKE image | size-phrase }
  [ BGCOLOR expression ]
  [ FGCOLOR expression ]
  [ CONVERT-3D-COLORS ]
  [ TOOLTIP tooltip ]
  [ STRETCH-TO-FIT [ RETAIN-SHAPE ] ] [ TRANSPARENT ]
```

`[ PRIVATE ] IMAGE image-name`

Defines and identifies an image widget as a data member for a class, and optionally specifies an access mode for that data member. Do not specify the access mode when defining an image widget for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

`IMAGE image-name`

Identifies the name by which the image widget is referenced. You can define the image widget in a procedure or a method within a class.

image-phrase

Specifies the file where the image is stored and the portion of the image to read. This is the syntax for image-phrase:

```
FILE name
  [ { IMAGE-SIZE | IMAGE-SIZE-CHARS | IMAGE-SIZE-PIXELS }
      width BY height
  ]
  [ FROM { { X n Y n } | { ROW n COLUMN n } } ]
```

For more information on this syntax, see the Image phrase reference entry.

You must specify either the LIKE option, an Image phrase or a Size phrase within the DEFINE IMAGE statement, and you may specify any two or all three.

LIKE *image*

Specifies a previously defined image from which this image inherits attributes. You can override specific attributes by specifying other options of the DEFINE IMAGE statement.

You must specify either the LIKE option, an Image phrase or a Size phrase within the DEFINE IMAGE statement, and you may specify any two or all three.

size-phrase

Specifies the outside dimensions of the image widget. This is the syntax for size-phrase:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

If you specify SIZE or SIZE-CHARS, the units are characters; if you specify SIZE-PIXELS, the units are pixels. If you use character units, the values *width* and *height* must be decimal constants; for pixel units, they must be integer constants. For more information, see the SIZE phrase reference entry.

You must specify either the LIKE option, an Image phrase or a Size phrase within the DEFINE IMAGE statement, and you may specify any two or all three.

BGCOLOR *expression*

Has no effect; supported only for backward compatibility.

FGCOLOR *expression*

Has no effect; supported only for backward compatibility.

CONVERT-3D-COLORS

Specifies that the colors associated with an image will be converted to the system 3D colors when an image is loaded. Table 25 describes the color conversion process.

Table 25: 3D-color conversions for images

If the color is ...	And the original Red-Green-Blue (RGB) color value is ...	Then the new converted system color is ...
White	(255, 255, 255)	System button highlight color.
Light Gray	(192, 192, 192)	System button face color.
Dark Gray	(128, 128, 128)	System button shadow color.
Black	(0, 0, 0)	System button text color.

During a session, if Windows notifies Progress that the system colors are changed, all images that have this option are reloaded and converted to the new system colors.

TOOLTIP *tooltip*

Allows you to define a help text message for an image widget. Progress automatically displays this text when the user pauses the mouse pointer over the image widget.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed from the button. No ToolTip is the default. ToolTips are supported in Windows only.

STRETCH-TO-FIT

Forces the image to expand or contract to fit within the image widget's boundaries.

This option has no effect if an icon is displayed on the image widget.

RETAIN-SHAPE

Indicates that the image should retain its aspect ratio (expand or contract equally in both dimensions). This may leave some uncovered space at the bottom or right of the image widget.

RETAIN-SHAPE is ignored if STRETCH-TO-FIT is FALSE or if an icon is displayed on the image widget.

TRANSPARENT

Indicates that the background color of the image is transparent. The background color is determined by the color of the pixel in the lower left corner of the image.

The TRANSPARENT option overrides the CONVERT-3D-COLORS option; if both are set, CONVERT-3D-COLORS is ignored.

This option has no effect if an icon is displayed on the image widget.

Example

This procedure defines an image widget named `trashcan`, and loads into the widget a series of operating system image files that create an animation of a fire burning in a trash can. The user begins the animation by choosing the Animate button. The procedure depends on the existence of image filenames `ANI01`, `ANI02`. ... `ANI14`.

r-image.p

```
DEFINE VARIABLE repeat_loop AS INTEGER.
DEFINE VARIABLE animation_loop AS INTEGER.
DEFINE VARIABLE ok AS LOGICAL.
DEFINE BUTTON animate LABEL "Animate".
DEFINE IMAGE trashcan FILE "ANI01.BMP".
DISPLAY animate trashcan WITH FRAME y TITLE "**  Animation Sample  **".

ON CHOOSE OF animate IN FRAME y DO:
  /* Begin Animation */
  DO repeat_loop = 1 TO 5:
    DO animation_loop = 1 TO 14:
      ok = trashcan:LOAD-IMAGE("ANI" + STRING(animation_loop,"99"))
        IN FRAME y.
    END.
  END.
END.
UPDATE animate WITH FRAME y.
```

Notes

- In Windows, if the file has no extension, Progress by default looks for image files with either a `.bmp` or `.ico` extension.
- To create the static image you are defining, you must define a static frame that contains the image. Each frame you define that contains the same image creates an additional instance of that image. The widget handle for a static image is not available until the image is created.
- Progress only performs the color conversion process on bitmaps (`.bmp` files) that contain 256 or fewer colors. However, you might consider using 16-color bitmaps because only the first sixteen entries in the bitmap's color table will be converted.
- Icon colors (`.ico` files) are not converted, even if CONVERT-3D-COLORS is TRUE.
- See Image phrase for the list of supported image file formats.
- You can specify an application-defined widget ID for a static image using the *`form-item`* phrase in either the FORM statement or the DEFINE FRAME statement. See the FORM statement and DEFINE FRAME statement reference entries for more information.

See also FORM statement, Image phrase

DEFINE MENU statement

Defines and creates a pop-up menu or a menu bar for use in one or more procedures.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ [ NEW ] SHARED ] [ PRIVATE ] MENU menu-name
  [ FGCOLOR expression ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ PFCOLOR expression ]
  [ FONT number ]
  [ { TITLE title } | MENUBAR ]
  [ { LIKE menu } | menu-element-descriptor ... ]
```

`NEW SHARED MENU` *menu-name*

Defines and identifies a menu that can be used by other procedures. The menu remains available to other procedures until the procedure that contains this statement ends.

`SHARED MENU` *menu-name*

Defines and identifies a menu that was created in another procedure with the DEFINE NEW SHARED MENU statement.

`[ PRIVATE ] MENU` *menu-name*

Defines and identifies a menu widget as a data member for a class, and optionally specifies an access mode for that data member. Do not specify the access mode when defining a menu widget for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

`MENU` *menu-name*

Identifies the name of the menu you are defining. You can define the menu widget in a procedure or a method within a class.

`BGCOLOR` *expression*

Supported only for backward compatibility. Progress does not support this option in Windows or character interfaces.

`DCOLOR` *expression*

Specifies the display color for the menu in character interfaces. This option is ignored in graphical interfaces.

`FGCOLOR` *expression*

Supported only for backward compatibility. Progress does not support this option in Windows or character interfaces.

`PFCOLOR` *expression*

Specifies the prompt-for color for the menu in character interfaces. This option is ignored in graphical interfaces.

`FONT` *number*

Supported only for backward compatibility. Progress does not support this option in Windows or character interfaces.

`MENUBAR`

Specifies that the menu displays as a menu bar.

`TITLE` *title*

Specifies the title of the menu. Only pop-up menus can have titles. This option is invalid for menu bars. The title displays at the top of the menu. In environments that do not support this option, it is ignored.

`LIKE` *menu*

Specifies a previously defined menu whose characteristics you want to apply to the new menu. If you name a menu with this option, you must have defined that menu previously in the procedure.

menu-element-descriptor

Specifies an element display on the menu. Each element is either a normal menu item, a submenu, a rule, or a blank space. The last two are valid only for pop-up menus. You must specify one or more menu elements, unless you use the LIKE option.

This is the syntax for *menu-element-descriptor*:

```
{ menu-item-phrase
   SUB-MENU submenu [ DISABLED ] [ LABEL label ]
   RULE
   SKIP
}
```

RULE

Specifies that a rule or line is inserted at this point in the menu. You can use this, for example, to divide the menu into sections.

SKIP

Specifies that a blank line is inserted at this point in the menu. You can use this, for example, to divide the menu into sections.

SUB-MENU *submenu* [DISABLED] [LABEL *label*]

Specifies that a submenu displays as a menu item. The submenu must be previously defined in the procedure. The submenu appears when the user chooses that item. The submenu cannot be a menu bar. The DISABLED and LABEL options for a submenu are the same as described for the *menu-item-phrase*.

menu-item-phrase

Specifies a normal menu item. This is the syntax for *menu-item-phrase*:

```
MENU-ITEM menu-item-name
  [ ACCELERATOR keylabel ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ DISABLED ]
  [ FGCOLOR expression ]
  [ FONT expression ]
  [ LABEL label ]
  [ PFCOLOR expression ]
  [ READ-ONLY ]
  [ TOGGLE-BOX ]
  [ trigger-phrase ]
```

MENU-ITEM *menu-item-name*

The name of the menu item you are defining.

ACCELERATOR *keylabel*

Specifies a keyboard accelerator for this menu item. A keyboard accelerator is a key—sometimes modified by **SHIFT**, **CONTROL**, or **ALT**—that chooses a menu item even if the menu is not displayed. The value *keylabel* must be character-string expression that evaluates to a valid key label recognized by Progress, such as a, **F1**, or **ALT+SHIFT+F1**.

BGCOLOR *expression*

Specifies the background color for the menu item in graphical environments. If you omit this option, the menu item inherits the background color of the menu.

DCOLOR *expression*

Specifies the display color for the menu item in character interfaces. If you omit this option, the menu item inherits the display color of the menu.

DISABLED

Specifies that the menu item is initially disabled for input. This means that the user cannot choose this item. Disabled items are grayed out (in environments that support it).

FGCOLOR *expression*

Specifies the foreground color for the menu item in graphical environments. If you omit this option, the menu item inherits the foreground color of the menu.

FONT *expression*

Specifies the font for the menu item. If you omit this option, the menu item inherits the font of the menu.

LABEL *label*

Specifies the text that is displayed in the menu for a choosable menu item or submenu. Include an ampersand (&) within the label to assign the following letter as a mnemonic for the menu item. This means that when the menu is displayed, the user can choose the item by pressing that single key. If you do not include an ampersand within the label, Windows treats the first character as a mnemonic.

To include a literal ampersand within a label, specify two ampersands (&&).

PFCOLOR *expression*

Specifies the prompt-for color for the menu item in character interfaces. If you omit this option, the menu item inherits the prompt-for color of the menu.

READ-ONLY

Specifies that this menu item is read-only text. The user cannot choose this item.

TOGGLE-BOX

Specifies that the menu item is displayed as a checkbox that the user can toggle on or off. In environments that do not support this option, it is ignored.

trigger-phrase

Specifies application triggers for the menu item. Typically, you associate a CHOOSE trigger with each menu item.

For more information, see the Trigger phrase reference entry.

Example

The `r-bar.p` procedure defines a menu bar, mbar, that contains three pull-down submenus labeled Topic, Move, and Exit. The handle of mbar is assigned to the current window. The ON statements define triggers to execute when you choose the corresponding menu items.

r-bar.p

```
DEFINE SUB-MENU topic
  MENU-ITEM numbr     LABEL "Cust. Number"
  MENU-ITEM addr      LABEL "Address"
  MENU-ITEM othrinfo  LABEL "Other".
DEFINE SUB-MENU move
  MENU-ITEM forward    LABEL "NextRec" ACCELERATOR "PAGE-DOWN"
  MENU-ITEM backward    LABEL "PrevRec" ACCELERATOR "PAGE-UP".
    DEFINE SUB-MENU quitit
  MENU-ITEM quititem   LABEL "E&xit".

DEFINE MENU mbar     MENUBAR
  SUB-MENU topic   LABEL "Topic"
  SUB-MENU move    LABEL "Move"
  SUB-MENU quitit  LABEL "E&xit".

ON CHOOSE OF MENU-ITEM numbr
  DISPLAY customer.cust-num.
ON CHOOSE OF MENU-ITEM addr
  DISPLAY customer.address customer.address2 customer.city
    customer.state customer.postal-code WITH FRAME addr-frame NO-LABELS
    COLUMN 25.
ON CHOOSE OF MENU-ITEM othrinfo
  DISPLAY customer EXCEPT name cust-num address
    address2 city state postal-code
    WITH FRAME oth-frame SIDE-LABELS.
ON CHOOSE OF MENU-ITEM forward
  DO:
    HIDE ALL NO-PAUSE.
    CLEAR FRAME name-frame.
    FIND NEXT customer NO-ERROR.
    IF AVAILABLE(customer)
    THEN DISPLAY customer.name WITH FRAME name-frame.
  END.
ON CHOOSE OF MENU-ITEM backward
  DO:
    HIDE ALL NO-PAUSE.
    CLEAR FRAME name-frame.
    FIND PREV customer NO-ERROR.
    IF AVAILABLE(customer)
    THEN DISPLAY customer.name WITH FRAME name-frame.
  END.

FIND FIRST customer.
DISPLAY customer.name LABEL "Customer Name" WITH FRAME name-frame.
ASSIGN CURRENT-WINDOW:MENUBAR = MENU mbar:HANDLE.
WAIT-FOR CHOOSE OF MENU-ITEM quititem.
```

Notes

- You cannot define a SHARED or NEW SHARED menu widget in a persistent procedure. If you do, Progress raises ERROR on the RUN statement that creates the procedure.
- You cannot define a SHARED or NEW SHARED menu widget in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- Keyboard accelerators are specified for menu-items forward and backward. The user can press PAGE-DOWN key to look at the next customer record and the PAGE-UP to view the previous customer record.
- The menu item quititem has a label E&xit; the ampersand makes X the mnemonic for that menu item.
- You cannot define a submenu with the same name more than once in the same menu tree. Thus, if menu mFile contains both submenu mOptions and submenu mSave, submenu mSave cannot also contain submenu mOptions.
- Menu items in different menus and submenus can have the same names. In the above procedure, the menu items in myfile and myobjects share the same names. To avoid ambiguity, use the IN MENU or IN SUB-MENU option to identify the parent menu or submenu.
- There are instances where you cannot avoid ambiguity in menu item references. In such instances, Progress always references the first unambiguous instance of the menu item. In particular, if the same submenu containing a menu item appears in more than one menu and each menu defines another instance of the same menu item, you can only reference that menu item in the submenu from the first menu that contains it. Thus, if submenu mOptions contains menu item mSave and the menus mFile and mDraw (in that order) both contain submenu mOptions and another menu item mSave, you can only reference menu item mSave in submenu mOptions from menu mFile. You cannot uniquely reference menu item mSave in submenu mOptions from menu mDraw because menu mDraw contains another menu item mSave. For more information on menu item references, see the chapter on menus in *OpenEdge Development: Progress 4GL Handbook*.

See also COLOR phrase, DEFINE SUB-MENU statement, RUN statement

DEFINE PARAMETER statement

Defines a run-time parameter in a Progress subprocedure, Windows dynamic link library (DLL) routine, UNIX shared library routine, or ActiveX control event procedure.

Note: To define run-time parameters of a user-defined function, or a method within a class (including constructor methods), see the Parameter definition syntax reference entry in this book.

Each parameter requires its own DEFINE statement. The parameters must be specified in the RUN statement in the same order they are defined with DEFINE statements. In addition, the parameter types (INPUT, OUTPUT, INPUT-OUTPUT, RETURN, TABLE, TABLE-HANDLE, DATASET, DATASET-HANDLE, and BUFFER) specified in the DEFINE and RUN statements must agree. The corresponding data types and run-time values must also be compatible enough to allow Progress to perform any necessary conversions.

Syntax

```
DEFINE { INPUT | OUTPUT | INPUT-OUTPUT | RETURN }
  PARAMETER parameter
  {  AS [ HANDLE TO ] datatype
   | AS [ CLASS ] { type-name }
   | LIKE field }
  [ EXTENT [ expression ] ]
  [ [ NOT ] CASE-SENSITIVE ]
  [ FORMAT string ]
  [ DECIMALS n ]
  [ INITIAL constant ]
  [ COLUMN-LABEL label ]
  [ LABEL string ]
  [ NO-UNDO ]
```

```
DEFINE PARAMETER BUFFER buffer FOR table
  [ PRESELECT ]
  [ LABEL label ]
```

```
DEFINE { INPUT | OUTPUT | INPUT-OUTPUT } PARAMETER
  {   TABLE FOR temp-table-name [ APPEND ] [ BIND ]
    | TABLE-HANDLE temp-table-handle [ APPEND ] [ BIND ]
    | DATASET FOR dataset-name [ APPEND ] [ BIND ]
    | DATASET-HANDLE dataset-handle [ APPEND ] [ BIND ]
  }
```

INPUT PARAMETER

Defines a parameter that gets its value from one of the following sources:

- If the calling procedure runs the current (called) procedure synchronously, the value comes from the corresponding INPUT parameter of the RUN statement.
- If the current procedure is the event procedure specified to handle the PROCEDURE-COMPLETE event for an asynchronous remote procedure, the value comes from the corresponding OUTPUT or INPUT-OUTPUT parameter of the remote procedure.

OUTPUT PARAMETER

Defines a parameter that returns a value to one of the following destinations:

- If the calling procedure runs the current (called) procedure synchronously, the value is returned to the corresponding OUTPUT parameter of the RUN statement in the calling procedure.
- If the calling procedure runs the current (called) procedure as an asynchronous remote procedure, the value is returned to the corresponding INPUT parameter of the event procedure specified to handle the PROCEDURE COMPLETE event for the current procedure.

`INPUT-OUTPUT PARAMETER`

Defines a parameter that receives an initial value passed from the calling procedure that can be subsequently modified by the called procedure. The calling procedure cannot pass a literal value. The called procedure returns the modified value to one of the following destinations:

- If the calling procedure runs the current (called) procedure synchronously, the value is returned to the corresponding INPUT-OUTPUT parameter of the RUN statement in the calling procedure.
- If the calling procedure runs the current (called) procedure as an asynchronous remote procedure, the value is returned to the corresponding INPUT parameter of the event procedure specified to handle the PROCEDURE COMPLETE event for the current procedure.

`RETURN PARAMETER`

Defines a parameter that holds the return value of a DLL or UNIX shared library routine. When the DLL routine returns, the value of this parameter is passed back to the calling procedure. You can only have one RETURN parameter per routine.

`parameter`

Identifies the name of the parameter you want to define.

`AS [ HANDLE TO ]` *`datatype`*

Specifies the data type of the parameter.

For Progress subprocedures, *`datatype`* can specify any built-in Progress data type used to define variables. For more information, see the reference entry for the DEFINE VARIABLE statement.

For DLL or UNIX shared library routines, *`datatype`* can specify a Progress DLL data type. Progress DLL data types include the built-in Progress data types CHARACTER and MEMPTR, Windows DLL-equivalent data types, and UNIX shared library data types.

Table 26 shows how Windows DLL and UNIX shared library data types map to Progress DLL data types.

Table 26: Data types for DLL and UNIX shared library routine parameters *(1 of 2)*

Example C data type	Progress DLL parameter data type	Windows DLL and UNIX shared library data type
`char`	`BYTE`	8-bit unsigned integer
`short`	`SHORT`	16-bit signed integer
`unsigned short`	`UNSIGNED-SHORT`	16-bit unsigned integer
`long, int`[1]	`LONG`[2]	32-bit signed integer
`float`	`FLOAT`	4-byte floating point
`double`	`DOUBLE`	8-byte floating point
`char*`	`CHARACTER`	Address (usually 32 bits)

Table 26: Data types for DLL and UNIX shared library routine parameters *(2 of 2)*

Example C data type	Progress DLL parameter data type	Windows DLL and UNIX shared library data type
c-data-type[3]	HANDLE TO *parameter-data-type*[3]	Address (usually 32 bits)
char*, *output-pointer* (which can be char**, short**, and so on), or a pointer to a structure.	MEMPTR	Address (usually 32 bits)

1 The C data type *int* generally specifies a size that depends on the operating system.

2 To pass a NULL pointer value to a DLL routine, pass 0 using a LONG parameter. Do not use a null MEMPTR variable to pass a NULL value. However, you can pass a NULL value in one or more elements of a MEMPTR array. If this conflicts with another way to call the DLL routine, specify a second declaration for the same routine using the ORDINAL option of the PROCEDURE statement.

3 You can use the HANDLE TO option to specify a pointer to a scalar type. Therefore, you can use the HANDLE TO option with the parameter data types (that is, BYTE, SHORT, UNSIGNED-SHORT, LONG, FLOAT, and DOUBLE) in order to specify a pointer to the respective C data types (that is, char, short, unsigned short, long, int, float, and double). For a CHARACTER or MEMPTR parameter, it is redundant because this data type is always passed using a pointer (*char*)*.

Caution: For CHARACTER parameters, Progress always passes the routine a pointer to the character or character string value rather than the value itself. If the routine modifies the value, it can also modify Progress memory outside the bounds of the CHARACTER value with unpredictable results. For this reason, Progress does not allow you to use OUTPUT or RETURN for CHARACTER or LONGCHAR parameters, as well as CHARACTER or LONGCHAR array parameters, and does not recommend you use INPUT-OUTPUT for CHARACTER or LONGCHAR parameters. Rather, pass the character string as a MEMPTR parameter. For more information, see *OpenEdge Development: Programming Interfaces*.

Note: You cannot use RETURN for any type of array parameter.

To indicate that the DLL or UNIX shared library parameter is a pointer to a value rather than the value itself, use the HANDLE option. The HANDLE option is required when the DLL routine expects a pointer to the value. Note that the CHARACTER data type implies the HANDLE option, whether or not you specify it. The TO keyword aids readability but has no meaning.

For ActiveX control event procedures, *datatype* can specify the built-in Progress data type that maps to the COM object data type of an ActiveX event parameter. Table 27 shows how the COM object data types for event parameters (shown as ActiveX data types) map to Progress data types.

Table 27: Data types for ActiveX control event procedures *(1 of 2)*

ActiveX data type[1]	Progress data type
Array	Progress array variable
Array of bytes	RAW
Boolean (2-byte integer)	LOGICAL
Byte (unsigned byte)	INTEGER
Currency (8-byte integer with fixed decimal point)	DECIMAL
Date Double	DATE DATETIME DATETIME-TZ
Decimal	DECIMAL
Double (8-byte floating point)	DECIMAL
Error Code	INTEGER
Integer (2-byte integer)	INTEGER
Long (4-byte integer)	INTEGER
Object (32-bit value)	COM-HANDLE
String (character string type)	CHARACTER LONGCHAR
Unsigned Byte	INTEGER

Table 27: Data types for ActiveX control event procedures *(2 of 2)*

ActiveX data type[1]	Progress data type
Unsigned Long (4-byte integer)	INTEGER
Unsigned Short (2-byte integer)	INTEGER
Variant (variable type)	<ANYTYPE>[2]

[1] For more information on these data type implementations for COM objects, see *OpenEdge Development: Programming Interfaces*.

[2] For Variant event parameters, the AppBuilder specifies <ANYTYPE> as a place holder. You must change <ANYTYPE> to the data type that most closely matches the expected value. For more information, see the available documentation on the event parameter.

`AS [ CLASS ] { type-name }`

Defines a class or interface parameter. Progress passes the object reference for a class or interface parameter (by value), not the class or interface itself.

type-name

A character string that specifies the type name of a class or interface. Specify a type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

If the specified class or interface type name conflicts with an abbreviation of a built-in Progress data type name, such as INT for INTEGER, you must specify the CLASS keyword.

`LIKE, CASE SENSITIVE, FORMAT, DECIMALS, INITIAL, COLUMN-LABEL, LABEL, NO-UNDO`

For descriptions of these options, see the DEFINE VARIABLE statement reference entry.

`EXTENT [ expression ]`

Defines a determinate array parameter (which has a defined number of elements) or an indeterminate array parameter (which has an undefined number of elements). To define a determinate array parameter, specify the EXTENT option with the *expression* argument. This optional argument evaluates to an integer value that represents an extent for the array parameter. To define an indeterminate array parameter, specify the EXTENT option without the *expression* argument. Progress determines the size of indeterminate array parameters at runtime.

You can pass a determinate array or indeterminate array to a procedure or function that declares the parameter as an indeterminate array. You cannot pass an indeterminate array to a procedure or function that declares the parameter as a determinate array. If you call a procedure or function that fixes the number of elements in an indeterminate array, Progress treats the fixed indeterminate array as a determinate array.

You cannot pass an indeterminate array to a COM object, DLL routine, or UNIX shared library routine.

If you want to define a parameter that is like an array variable or field, using the LIKE option, but you do not want the parameter to be an array, you can use EXTENT 0 to indicate a non-array parameter.

If you are using the AS *datatype* option and you do not use the EXTENT option (or you specify *expression* as 0), the parameter is not an array parameter. If you are using the LIKE *field* option and you do not use the EXTENT option, the parameter uses the extent defined for the database field you name (if any).

PARAMETER BUFFER *buffer* FOR *table* [PRESELECT] [LABEL *label*]

Defines a buffer parameter. You can pass a buffer associated with a database table to a buffer parameter. You cannot pass a work table to a buffer parameter. A buffer parameter is always INPUT-OUTPUT. You cannot pass buffer parameters to the AppServer.

For descriptions of the PRESELECT and LABEL options, see the DEFINE BUFFER statement reference entry.

TABLE FOR *temp-table-name*

Defines a temporary table parameter.

You can pass a temporary table parameter to both local and remote procedures. Progress passes the parameter by value, by default. That is, the calling routine and the called routine each have their own instance of the temporary table. When you invoke the RUN statement, Progress deep-copies the temporary table from one routine's instance to the other routine's instance. Which table travels depends on whether the parameter is INPUT, OUTPUT, or INPUT-OUTPUT. When you pass a temporary table as an INPUT parameter, Progress overlays the stationary instance with the traveling table, by default. You can also append the traveling table to the end of the stationary instance by specifying the APPEND option. For more information about the APPEND option, see the option description below.

When passing a temporary table parameter to a local procedure, you can override the default by passing the parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option).

Passing a temporary table parameter by reference or by binding allows the calling routine and the called routine to access the same object instance (instead of deep-copying the parameter).

Note: When you specify the BIND option in the DEFINE PARAMETER statement, you must also specify the BIND option in the RUN statement.

For more information about passing a temporary table parameter by reference or by binding, see the Parameter passing syntax reference entry in this book. For more information about temporary table parameters, see *OpenEdge Development: Progress 4GL Handbook*.

`TABLE-HANDLE` *`temp-table-handle`*

Defines a temporary table handle parameter.

`DATASET` *`dataset-name`*

Defines a static ProDataSet object parameter.

You can pass a ProDataSet object parameter to both local and remote procedures. Progress passes the parameter by value, by default. That is, the calling routine and the called routine each have their own instance of the object. When you invoke the RUN statement, Progress deep-copies the object from one routine's instance to the other routine's instance. Which object travels depends on whether the parameter is INPUT, OUTPUT, or INPUT-OUTPUT. When you pass a ProDataSet object as an INPUT parameter, Progress overlays the stationary instance with the traveling ProDataSet object, by default. You can also append the traveling ProDataSet object to the end of the stationary instance by specifying the APPEND option. For more information about the APPEND option, see the option description below.

When passing a ProDataSet object parameter to a local procedure, you can override the default by passing the parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). Passing a ProDataSet object parameter by reference or by binding allows the calling routine and the called routine to access the same object instance (instead of deep-copying the parameter).

Note: When you specify the BIND option in the DEFINE PARAMETER statement, you must also specify the BIND option in the RUN statement.

For more information about passing a ProDataSet object parameter by reference or by binding, see the Parameter passing syntax reference entry in this book. For more information on ProDataSet object parameters, see *OpenEdge Development: Progress 4GL Handbook*.

`DATASET-HANDLE` *`dataset-handle`*

Defines a ProDataSet object handle parameter.

`FOR . . . APPEND`

Specifies whether or not to append the traveling temporary table data to the stationary temporary table instance. To append input parameter data, specify the APPEND option in the DEFINE PARAMETER statement. To append output parameter data, specify the APPEND option in the RUN statement.

For *`temp-table-name`*, FOR is required and APPEND is optional.

For *`temp-table-handle`*, both FOR and APPEND are optional and are used together. Without the FOR keyword, the temporary table handle is local to the called procedure - it is created when the procedure starts and deleted when the procedure completes. Therefore, there is nothing to APPEND the received parameter to. When the FOR keyword is used, the temporary table handle is defined at a higher level outside the called procedure and continues to exist after the called procedure completes.

`BIND`

Indicates that a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter binds a reference-only object in one routine to an object instance defined and instantiated in another local routine.

When you define a reference-only object in the calling routine, and you want to bind that object definition to an object instance in the called routine, define the parameter by specifying the BIND option in an INPUT or INPUT-OUTPUT parameter definition. When you define a reference-only object in the called routine, and you want to bind that object definition to an object instance in the calling routine, define the parameter by specifying the BIND option in an OUTPUT parameter definition. In either case, the reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates.

Caution: Do not delete the object or routine to which a reference-only object is bound, or you might be left with references to an object that no longer exists.

You can bind multiple reference-only object definitions to the same object instance. You can also bind a single reference-only object definition to the same object instance multiple times without generating an error. However, you cannot bind a single reference-only object definition to multiple object instances.

When passing one of these parameters to a remote procedure, Progress ignores the BIND option and deep-copies the parameter based on the specified parameter mode.

For more information about passing these parameters by binding, see the Parameter passing syntax reference entry in this book.

Examples

In the following examples, the `r-runpar.p` procedure runs a subprocedure called `r-param.p` and passes the subprocedure an INPUT parameter. The subprocedure `r-param.p` displays the INPUT parameter.

r-runpar.p

```
RUN r-param.p (INPUT 10).
```

r-param.p

```
DEFINE INPUT PARAMETER int-param AS INTEGER.
DISPLAY int-param LABEL "Integer input param"
  WITH SIDE-LABELS.
```

In the following example, the `r-runpr1.p` procedure runs a subprocedure called `r-param1.p`. This example illustrates the use of multiple parameters and shows that the parameters must be passed in the proper order and must be of the same data type. Note that if you do not specify a parameter type in the RUN statement, Progress assumes it is an input parameter.

r-runpr1.p

```
DEFINE VARIABLE new-param AS CHARACTER FORMAT "x(20)".
DEFINE VARIABLE out-param AS DECIMAL.
DEFINE VARIABLE in-param AS INTEGER INIT 20.
RUN r-param1.p (OUTPUT out-param, 10, OUTPUT new-param, in-param).
DISPLAY out-param LABEL "Updated YTD Sales" SKIP
    new-param LABEL "Status"
    WITH SIDE-LABELS.
```

r-param1.p

```
DEFINE OUTPUT PARAMETER xout-param AS DECIMAL.
DEFINE INPUT PARAMETER newin AS INTEGER.
DEFINE OUTPUT PARAMETER xnew-param AS CHARACTER.
DEFINE INPUT PARAMETER xin-param AS INTEGER.

FOR EACH customer:
   xout-param = balance + xout-param.
END.

DISPLAY xout-param LABEL "Balance" WITH SIDE-LABELS.
xout-param = xout-param + newin + xin-param.
xnew-param = "Example Complete".
```

In the following example, the `r-runpr2.p` procedure displays information from a database table and assigns the value of a database field to a variable called io-param. The variable is passed as an INPUT-OUTPUT parameter to a subprocedure called `r-param2.p`. The subprocedure `r-param2.p` performs a calculation on the INPUT-OUTPUT parameter, then passes it back to the main procedure. The `r-runpr2.p` assigns the value io-param to a database field, then displays io-param.

r-runpr2.p

```
DEFINE VARIABLE io-param AS INTEGER.

FOR EACH item:
  DISPLAY Item-name On-hand WITH 1 DOWN.
  io-param = Item.On-hand.
  RUN r-param2.p (INPUT-OUTPUT io-param).
  Item.On-hand = io-param.
  DISPLAY io-param LABEL "New Quantity On-hand".
END.
```

r-param2.p

```
DEFINE INPUT-OUTPUT PARAMETER io-param AS INTEGER.
DEFINE VARIABLE inp-qty AS INTEGER.
PROMPT-FOR inp-qty LABEL "Quantity Received?".
ASSIGN inp-qty.
io-param = io-param + inp-qty.
```

The following example uses a buffer parameter. The procedure `r-bufp.p` passes the Customer buffer to `r-fincus.p`. The `r-fincus.p` procedure then attempts to find a record using that buffer.

r-bufp.p

```
DEFINE BUTTON find-butt LABEL "Find Customer".

ENABLE find-butt Customer.Cust-num WITH FRAME cust-frame.

ON CHOOSE OF find-butt
  DO:
     RUN r-fincus.p(INPUT Customer.Cust-num:HANDLE IN FRAME cust-frame,
                    BUFFER Customer).
     DISPLAY Customer WITH FRAME cust-frame.
  END.

ON ENTRY OF Customer.Cust-num
  HIDE MESSAGE.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

r-fincus.p

```
DEFINE INPUT PARAMETER  wid-hand  AS WIDGET-HANDLE.
DEFINE PARAMETER BUFFER curr-buff FOR Customer.

FIND curr-buff WHERE curr-buff.Cust-num =
                     INT(wid-hand:SCREEN-VALUE) NO-ERROR.

IF NOT AVAILABLE(curr-buff)
THEN DO:
   MESSAGE "Record not found.".
   RETURN ERROR.
END.

RETURN.
```

The following example defines parameters for the DLL routine, `MessageBox()`, which displays a message on the screen:

r-dllex1.p

```
DEFINE VARIABLE result AS INTEGER.

MESSAGE "  It's a whole new world!"
  VIEW-AS
    ALERT-BOX MESSAGE
    BUTTONS OK
    TITLE "Progress Message".

RUN MessageBoxA (0, "  It's a whole new world, again!!",
      "Progress DLL access", 0, OUTPUT result).

PROCEDURE MessageBoxA EXTERNAL "user32.dll":
  DEFINE INPUT PARAMETER hwnd AS LONG.
  DEFINE INPUT PARAMETER mbtext AS CHARACTER.
  DEFINE INPUT PARAMETER mbtitle AS CHARACTER.
  DEFINE INPUT PARAMETER style AS LONG.
  DEFINE RETURN PARAMETER result AS LONG.
END.
```

Notes

- All procedure parameters are normally passed by value, by default. This means that for any INPUT-OUTPUT or OUTPUT parameter, the field or variable that receives the output value is not set by the called procedure until the procedure returns without error. An exception is made for local DATASET, DATASET-HANDLE, TABLE, and TABLE-HANDLE parameters, which you may pass by reference or by binding by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option. If you specify the BIND option in the RUN statement, you must also specify the BIND option in the DEFINE PARAMETER statement.

  For more information about passing parameters by reference or by binding, see the Parameter passing syntax reference entry in this book. For more information about passing parameters to both local and remote procedures, see *OpenEdge Development: Progress 4GL Handbook*.

- You cannot pass a BLOB or CLOB field as a parameter. To pass a BLOB or CLOB field as a parameter, you must include the field in a temp-table or convert the field to its MEMPTR or LONGCHAR counterpart, respectively.

- Buffer parameters are scoped in the same way as shared buffers. They also affect cursors defined in the calling procedure in the same way as shared buffers.

- For DLL or UNIX shared library routine declarations:
  - The LIKE *field* option can only specify a database field of type CHARACTER or a variable of type CHARACTER or MEMPTR.
  - The COLUMN, COLUMN-LABEL, DECIMALS, INITIAL, FORMAT, LABEL, and NO-UNDO options have no effect.
- You cannot pass a DATE, DATETIME, or DATETIME-TZ as a parameter to or from a DLL routine or a UNIX shared library routine.
- You can pass a LONGCHAR as a parameter to a DLL routine or a UNIX shared library routine. When passing a LONGCHAR parameter, Progress passes only the text string (not the code page information). You are responsible for setting the code page of a LONGCHAR parameter.
- You can pass an array of type INTEGER or DECIMAL as a parameter to or from a DLL routine or a UNIX shared library routine. You can pass an array of type CHARACTER to (not from) a DLL routine or a UNIX shared library routine.
- You cannot pass a variable or array that contains the Unknown value (?) to a DLL.
- RETURN parameters are supported only for DLL or UNIX shared library routines. The RETURN parameter type must match the OUTPUT parameter that returns the DLL function value in the RUN statement for the routine. You cannot pass an array as a RETURN parameter to DLL or UNIX shared library routines. Use a MEMPTR instead.
- If you specify a RETURN parameter as MEMPTR to return a character string, use the GET-STRING function to extract the CHARACTER value.
- For more information on DLL routine parameters and how they map to Progress data types, see the chapter on DLLs in *OpenEdge Development: Programming Interfaces*.
- You cannot pass a MEMPTR as a parameter to or from a COM object.

- You can pass a LONGCHAR, a DATETIME, a DATETIME-TZ, and an array as a parameter to or from a COM object.

  When passing a LONGCHAR parameter to a COM object, Progress passes only the text string (not the code page information). When receiving a text string from a COM object into a LONGCHAR parameter, Progress converts the text string to the code page associated with the LONGCHAR parameter only if the LONGCHAR has a fixed code page. Otherwise, Progress sets the LONGCHAR code page to UTF-16. If Progress cannot convert a LONGCHAR, it raises a runtime error.

  When passing a DATETIME or DATETIME-TZ parameter to a COM object, Progress represents the time to the millisecond. When passing a DATETIME-TZ parameter to a COM object, Progress first converts the DATETIME-TZ value relative to the local session's date and time, and then drops the time zone.

  When receiving a date from a COM object into a DATETIME or DATETIME-TZ parameter, Progress represents the time to the millisecond. When receiving a date from a COM object into a DATETIME-TZ parameter, Progress sets the time zone to the local session's time zone.

- For more information on ActiveX event parameters, or using COM objects in the 4GL, see the chapter in *OpenEdge Development: Programming Interfaces*.

- For dynamic temp-table parameters:

  – If the parameter is INPUT TABLE-HANDLE, the temp-table definition behind the handle plus the temp-table contents are sent from the caller to the called routine. The called routine may have either the dynamic INPUT TABLE-HANDLE or the static INPUT TABLE as a matching parameter.

  – If the parameter is INPUT TABLE-HANDLE, a new instance of the temp-table is created along with its handle, completely separate from the caller’s table, and is populated with the contents from the caller’s table.

    You can override this default behavior to allow the calling procedure and the called procedure to access the same object instance by passing the TABLE-HANDLE parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). If you specify the BIND option in the RUN statement, you must also specify the BIND option in the DEFINE PARAMETER statement.

  - If the parameter is OUTPUT TABLE-HANDLE, the handle plus the definition behind the handle are sent from the caller to the called routine. The called routine may have either the dynamic OUTPUT TABLE-HANDLE or the static OUTPUT TABLE as a matching parameter.
  - If the parameter is OUTPUT TABLE-HANDLE, and the handle is the Unknown value (?), no definition is sent to the called routine.
  - If the parameter is OUTPUT TABLE-HANDLE, the called routine sends back the definition behind the handle along with the contents of the output temp-table. In the caller, if the original handle was the Unknown value (?), a new instance of the temp-table is created and populated with the output contents. If the original handle is not the Unknown value (?), the caller's existing table must match the table being received from the called routine.
  - If the APPEND option is used, the new data is added to the existing table's data.
  - If the parameter is INPUT-OUTPUT TABLE-HANDLE, a combination of the above occurs.
  - If you call a remote procedure asynchronously (using the ASYNCHRONOUS option of the RUN statement) and pass a parameter as OUTPUT TABLE-HANDLE *temp-table-handle* APPEND, the event procedure must specify a corresponding DEFINE INPUT PARAMETER TABLE-HANDLE FOR *temp-table-handle* APPEND statement, and *temp-table-handle* must be global to both the calling procedure and the event procedure.

- If you define an INPUT TABLE parameter for an asynchronous event procedure with a data type that is different from the data type of the corresponding OUTPUT TABLE parameter passed from the AppServer, any failure to convert the passed value causes the event procedure to fail and Progress to display an error message on the client.
- You cannot specify a ProDataSet object or ProDataSet object handle as a parameter for an asynchronous remote procedure.
- For more information on working with asynchronous remote procedures and event procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

- For dynamic ProDataSet object parameters:
  - If the parameter is INPUT DATASET-HANDLE, the ProDataSet object definition behind the handle plus the ProDataSet object contents are sent from the caller to the called routine. The called routine may have either the dynamic INPUT DATASET-HANDLE or the static INPUT DATASET as a matching parameter.
  - If the parameter is INPUT DATASET-HANDLE, a new instance of the ProDataSet object is created along with its handle, completely separate from the caller's table, and is populated with the contents from the caller's table.

    You can override this default behavior to allow the calling procedure and the called procedure to access the same object instance by passing the DATASET-HANDLE parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). If you specify the BIND option in the RUN statement, you must also specify the BIND option in the DEFINE PARAMETER statement.
  - If the parameter is OUTPUT DATASET-HANDLE, the handle plus the definition behind the handle are sent from the caller to the called routine. The called routine may have either the dynamic OUTPUT DATASET-HANDLE or the static OUTPUT DATASET as a matching parameter.

    If the parameter is OUTPUT DATASET-HANDLE, and the caller's handle is the Unknown value (?), no definition is sent to the called routine.
  - If the parameter is OUTPUT DATASET-HANDLE, the called routine sends back the definition behind the handle along with the contents of the output ProDataSet object. In the caller, if the original handle is the Unknown value (?), a new instance of the ProDataSet object is created and populated with the output contents. If the original handle is not the Unknown value (?), the caller's existing object must match the object being received from the called routine.
  - If the APPEND option is used, the new data is added to the existing object's data.
  - If the parameter is INPUT-OUTPUT DATASET-HANDLE, a combination of the above occurs.

See also DEFINE BUFFER statement, DEFINE VARIABLE statement, DELETE PROCEDURE statement, RUN statement

DEFINE QUERY statement

Defines a query that can be opened with an OPEN QUERY statement and from which records can be retrieved with a GET statement or BROWSE widget.

Syntax

```
DEFINE [ [ NEW ] SHARED ] [ PRIVATE | PROTECTED ] QUERY query
  FOR bufname [ field-list ] [ , bufname [ field-list ] ] ...
  [ CACHE n ]
  [ SCROLLING ]
  [ RCODE-INFORMATION ]
```

`NEW SHARED QUERY` *query*

Defines and identifies a query to be shared with one or more procedures called directly or indirectly by the current procedure. The called procedures must define the same query name as SHARED. For shared queries, each *bufname* must be the name of a shared buffer. The shared buffers must be specified in the same order both across shared queries and in the OPEN QUERY.

`SHARED QUERY` *query*

Defines and identifies a query that was initially defined by another procedure as NEW SHARED. For shared queries, each *bufname* must be the name of a shared buffer. The shared buffers must be specified in the same order across shared queries and in the OPEN QUERY.

`[ PRIVATE | PROTECTED ] QUERY` *query*

Defines and identifies a query as a data member for a class, and optionally specifies an access mode for that data member. Do not specify an access mode when defining a query for a method within a class.

PRIVATE data members can be accessed only by the defining class. PROTECTED data members can be accessed by the defining class and any of its inheriting classes. The default access mode is PRIVATE.

Note: These options are applicable only when defining a data member for a class in a class definition (`.cls`) file.

`QUERY query`

Identifies the name of the query you are defining. You can define the query in a procedure or a method within a class.

`FOR bufname [ field-list ] [ , bufname [ field-list ] ] ...`

Specifies the buffers to be used by the query, where *bufname* is a table or alternate buffer name. For a shared query, each *bufname* must be a shared buffer. If the query definition references more than one buffer, it defines a join.

Once the query has been defined, you cannot change the buffers that it references, even if the query is closed and re-opened. For example, a buffer, buff1, is created for the customer table in a DEFINE QUERY or OPEN QUERY for the query, qry1. The query is run and closed. You cannot now DEFINE or OPEN qry1 with buff1 for the item table. You can reuse buffers with CREATE QUERY, but you must re-run QUERY-PREPARE.

The *field-list* specifies a list of fields to include or exclude when you open the query. This is the syntax for *field-list*:

```
{
     FIELDS [ ( [ field ... ] ) ]
   | EXCEPT [ ( [ field ... ] ) ]
}
```

The FIELDS option specifies the fields you want to include in the query, and the EXCEPT option specifies the fields that you want to exclude from the query. The *field* parameter is the name of a single field in the table specified by *bufname*. If *field* is an array reference, the whole array is retrieved even if only one element is specified. Specifying FIELDS with no *field* references causes Progress to retrieve sufficient information to extract the ROWID value for each record in the query (returnable using the ROWID function). Specifying EXCEPT with no *field* references or specifying *bufname* without a *field-list* causes Progress to retrieve all fields for each record in the query.

This statement defines a query to retrieve only the name and balance fields from the customer table:

```
DEFINE QUERY custq FOR customer FIELDS (name balance).
```

This statement defines a query to retrieve all fields of the customer table except the name and balance fields:

```
DEFINE QUERY custq FOR customer EXCEPT (name balance).
```

When you specify a field list for a query, Progress might retrieve additional fields or complete records depending on the type of query operation and the DataServer that provides the records. Thus, Progress:

- Retrieves any additional fields required by the client to complete the record selection.
- Retrieves complete records when you open the query with EXCLUSIVE-LOCK or update any row (such as with a browse). This ensures proper operation of updates and the local before-image (BI) file. For information on the local BI file, see *OpenEdge Data Management: Database Administration*.
- Retrieves complete records for DataServers that do not support SHARE-LOCK. For more information, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

Note: Always specify fields that you plan to reference in the field list. Only those extra fields that the client requires for record selection are added to the specified field list. Progress distributes record selection between the client and server depending on a number of factors that change with each OpenEdge release. Therefore, never rely on fields that you did not specify but which Progress fetches for its own needs; they might not always be available. There is no additional cost to specify a field in the list that you otherwise expect Progress to provide.

This query example retrieves the customer.cust-num field in addition to those specified in the field lists because it is required to satisfy the inner join between the customer and order tables.

r-qryjoin.p

```
DEFINE QUERY coq FOR customer FIELDS (name),
                     order FIELDS (order-num sales-rep).

OPEN QUERY coq FOR EACH customer, EACH order OF customer.

REPEAT:
    GET NEXT coq.
    IF NOT AVAILABLE (order) THEN LEAVE.
    DISPLAY customer.name customer.cust-num
            order.order-num order.sales-rep
    WITH FRAME q-frame 10 DOWN.
END.
```

However, do not rely on Progress to always provide such extra fields. For reliability, add the `cust-num` field to the customer field list. For example:

```
DEFINE QUERY coq FOR customer FIELDS (name cust-num),
                     order FIELDS (order-num sales-rep).
                        .
                        .
                        .
```

When you specify a field list in a shared query, you must specify the complete field list in the NEW SHARED query definition. Each corresponding SHARED query definition in another procedure file (.p) requires only the FIELDS or EXCEPT keywords, but can also include empty parentheses or the complete field list with no difference in functionality.

You can match this NEW SHARED query definition for customer with any of the following SHARED query definitions with no effective difference:

```
/* main.p */

DEFINE NEW SHARED QUERY q FOR customer FIELDS (cust-num name).
                          .
                          .
                          .
/* shared.p's */

DEFINE SHARED QUERY q FOR customer FIELDS.
DEFINE SHARED QUERY q FOR customer EXCEPT.
DEFINE SHARED QUERY q FOR customer FIELDS ().
DEFINE SHARED QUERY q FOR customer FIELDS (cust-num name).
```

If you define a NEW SHARED query with a field list and a matching SHARED query without a field list, or if you define a NEW SHARED query without a field list and a matching SHARED query with a field list, Progress raises the ERROR condition when you run the procedure file that contains the SHARED query.

CACHE *n*

Specifies the number of records of the query to hold in memory for a NO-LOCK query. Generally, caching more records produces better browse performance when accessing a database across a network. However, caching consumes both memory and CPU time for buffer management.

If you specify the CACHE option, the SCROLLING option is assumed. If a query is referenced in a DEFINE BROWSE statement, caching occurs by default. The default for a query involving only one table is 50 records. The default for a multi-table query is 30 records. If you specify CACHE 0 in the DEFINE QUERY statement, no caching occurs.

SCROLLING

Specifies that you can jump to a location within the list of records that satisfy the query by using the REPOSITION statement. If you do not use this option, you can use only the FIRST, NEXT, LAST, and PREV options of the GET statement to navigate within the list. Queries are faster if you do not use this option, but you must specify it to use the REPOSITION statement. For non-OpenEdge databases, if you do not specify SCROLLING, you can only move forward through the list of records using the FIRST and NEXT options of the GET statement.

RCODE-INFORMATION

Note: This option is the default behavior (thus, it has no effect). It is supported only for backward compatibility.

Examples

The following example defines two queries, q-salesrep and q-cust. The first is opened in the main procedure block and is used to find all salesrep records. The q-cust query is used to find all customers associated with a salesrep. The results of the q-cust query are displayed in a browse widget. The q-cust query is reopened each time you find a new salesrep.

r-defqry.p

```
DEFINE QUERY q-salesrep FOR salesrep
    FIELDS (salesrep.sales-rep salesrep.rep-name salesrep.region
            salesrep.month-quota).
DEFINE QUERY q-cust     FOR customer
    FIELDS (customer.cust-num customer.name customer.phone).

DEFINE BROWSE cust-brws QUERY q-cust
  DISPLAY customer.cust-num customer.name customer.phone
    WITH 5 DOWN TITLE "Customer Information".

DEFINE BUTTON b_next LABEL "Next".
DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.

FORM
   salesrep.sales-rep salesrep.rep-name salesrep.region
   salesrep.month-quota
   WITH FRAME rep-info SIDE-LABELS TITLE "Sales Rep. Info".

FORM b_next space(5) b_quit
  WITH FRAME butt-frame COLUMN 60.

ON CHOOSE OF b_next
   DO:
      GET NEXT q-salesrep.
      IF NOT AVAILABLE(salesrep) THEN GET FIRST q-salesrep.
      RUN disp-rep.
   END.

OPEN QUERY q-salesrep FOR EACH salesrep NO-LOCK.

GET FIRST q-salesrep.
RUN disp-rep.

ENABLE cust-brws WITH FRAME cust-info.
ENABLE ALL WITH FRAME butt-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.


PROCEDURE disp-rep.
      DISPLAY salesrep.sales-rep
          salesrep.rep-name
          salesrep.region
          salesrep.month-quota
      WITH FRAME rep-info CENTERED SIDE-LABELS TITLE "Sales Rep. Info".
      OPEN QUERY q-cust FOR EACH customer OF salesrep NO-LOCK.
END PROCEDURE.
```

The following example uses the RCODE-INFORMATION option of the DEFINE QUERY statement to extracts index information from a static query. If you run the example with the RCODE-INFORMATION option commented out, Progress reports a run time error.

r-rcdinf.p

```
/* r-rcdinf.p */
/* Extracts index information from a static query.*/

DEFINE QUERY q FOR customer RCODE-INFORMATION.
DEFINE VARIABLE h AS HANDLE.

h = QUERY q:HANDLE.
OPEN QUERY q FOR EACH cust BY name.
MESSAGE h:INDEX-INFORMATION.
```

Notes

- You cannot define a SHARED or NEW SHARED query in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- After you define a query, you must open it with the OPEN QUERY statement before you can fetch any records.
- A SHARED query remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the query as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED query, Progress includes the persistent procedure in the resolution of the corresponding NEW SHARED query as though the procedure were on the procedure call stack.
- Specifying a field list (*field-list*) for *bufname* can increase the performance of remote (network) queries substantially over specifying *bufname* alone. For more information, see *OpenEdge Development: Progress 4GL Handbook*.

- If you reference an unfetched database field in a query at run time, Progress raises the ERROR condition. Progress does not perform a compile-time check to ensure that the field is fetched because the compiler cannot reliably determine how a particular record will be read (that is, whether it is retrieved using a FIND statement, retrieved with or without a field list, including additional fields to satisfy join conditions, etc.).
- Unlike with block record retrieval operations that include record updates and deletes (FOR EACH, etc.), field lists generally enhance query performance even for queries whose rows you plan to update. Queries generate complete result lists, with or without field lists, before any updates to individual rows are applied.
- You can specify the Field List Disable (`-fldisable`) startup parameter to cancel field list retrieval and force Progress to retrieve complete records. This is a run-time client session parameter that is especially useful for deployed applications whose database triggers are later redefined to reference unfetched fields (raising the ERROR condition). Using `-fldisable` provides a workaround that allows the application to run (although more slowly) until the application can be fixed.
- You cannot specify field lists in an OPEN QUERY statement.
- In a shared query, the shared buffers must be specified in the same order across all the shared queries and in the OPEN QUERY statement.

See also CLOSE QUERY statement, CURRENT-RESULT-ROW function, DEFINE BROWSE statement, GET statement, NUM-RESULTS function, OPEN QUERY statement, REPOSITION statement, RUN statement

DEFINE RECTANGLE statement

Defines a rectangle widget for use in the current procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ PRIVATE ] RECTANGLE rectangle [ LIKE rectangle2 ]
  [ NO-FILL ]
  [ { EDGE-CHARS width } | { EDGE-PIXELS width } ]
  [ DCOLOR expression ]
  [ BGCOLOR expression ]
  [ FGCOLOR expression ]
  [ GRAPHIC-EDGE ]
  [ PFCOLOR expression ]
  [ size-phrase ]
  [ TOOLTIP tooltip ]
  { [ trigger-phrase ] }
```

[PRIVATE] RECTANGLE *rectangle*

Defines and identifies a rectangle widget as a data member for a class, and optionally specifies an access mode for that data member. Do not specify the access mode when defining a rectangle widget for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

RECTANGLE *rectangle*

Identifies the name of the rectangle widget you are defining. You can define the rectangle widget in a procedure or a method within a class.

LIKE *rectangle2*

Specifies a previously defined rectangle whose characteristics you want to apply to the new rectangle. If you name a rectangle with this option, you must have defined that rectangle previously in the procedure.

NO-FILL

Indicates that only the outline of the rectangle should be drawn. By default, the rectangle is filled with the background color.

EDGE-CHARS *width*

Specifies the width of the rectangle outline in characters. The default width is 1. If you do not want an edge on the rectangle, specify EDGE-CHARS 0.

EDGE-PIXELS *width*

Specifies the width of the rectangle outline in pixels. The default width is 1. If you do not want an edge on the rectangle, specify EDGE-PIXELS 0.

DCOLOR *expression*

Specifies the fill color of the rectangle in character interfaces. This option is ignored in graphical interfaces.

BGCOLOR *expression*

Specifies the background color or fill color of the rectangle in graphical interfaces. This option is ignored in character interfaces.

FGCOLOR *expression*

Specifies the foreground color or edge color of the rectangle in graphical interfaces. This option is ignored in character interfaces.

GRAPHIC-EDGE

Specifies that in a character interface, the rectangle is drawn with graphic characters. This option is ignored in a graphical interface. This overrides the EDGE-CHARS and EDGE-PIXELS options. The border is one graphic unit thick.

PFCOLOR *expression*

Specifies the edge color of the rectangle in character interfaces. This option is ignored in graphical interfaces. It is also ignored if you specify GRAPHIC-EDGE.

size-phrase

Specifies the outside dimensions of the rectangle widget. This is the syntax for *size-phrase*:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

If you specify SIZE or SIZE-CHARS, the units are characters; if you specify SIZE-PIXELS, the units are pixels. For character units, the values *width* and *height* must be decimal constants. For pixels units, they must be integer constants. For more information, see the SIZE phrase reference entry.

TOOLTIP *tooltip*

Allows you to define a help text message for a rectangle widget. Progress automatically displays this text when the user pauses the mouse button over the rectangle widget.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed. No ToolTip is the default. The TOOLTIP option is supported in Windows only.

trigger-phrase

Specifies application triggers for the rectangle.

For more information, see the Trigger phrase reference entry.

Example

The following example uses a set of thin rectangles as lines to create graphic columns within a frame background:

r-bkgrnd.p

```
DEFINE VARIABLE item-tot AS DECIMAL LABEL "Value" NO-UNDO.

DEFINE RECTANGLE vline1 SIZE .4 BY 5 EDGE-PIXELS 2.
DEFINE RECTANGLE vline2 LIKE vline1.
DEFINE RECTANGLE vline3 LIKE vline1.
DEFINE RECTANGLE vline4 LIKE vline1.
DEFINE RECTANGLE vline5 LIKE vline1.
DEFINE RECTANGLE vline6 LIKE vline1.

DEFINE RECTANGLE hline SIZE 78 BY .1 EDGE-PIXELS 2.

DEFINE FRAME item-info
  item.item-num
  item.item-name
  item.on-hand
  item.re-order
  item.on-order
  item.price
  item-tot
  BACKGROUND skip(1) hline
    vline1 AT 9
    vline2 AT 25
    vline3 AT 33
    vline4 AT 42
    vline5 AT 51
    vline6 AT 65
  WITH TITLE "Inventory Current Value" CENTERED USE-TEXT 5 DOWN.

FOR EACH item NO-LOCK WITH FRAME item-info:
  DISPLAY item.item-num
    item.item-name
    item.on-hand
    item.re-order
    item.on-order
    item.price
    item.on-hand * item.price @ item-tot.
```

Notes

- To create the static rectangle you are defining, you must define a static frame that contains the rectangle. Each frame you define that contains the same rectangle creates an additional instance of that rectangle. The widget handle for a static rectangle is not available until the rectangle is created.
- You can specify an application-defined widget ID for a static rectangle widget using the *form-item* phrase in either the FORM statement or the DEFINE FRAME statement. See the FORM statement and DEFINE FRAME statement reference entries for more information.
- When defining a rectangle, you must specify either the LIKE option or the size phrase.

See also FORM statement

DEFINE STREAM statement

Use the DEFINE STREAM statement when you want to use streams other than the two unnamed streams supplied by Progress. Using other streams let you, in a single procedure, get input from more than one source simultaneously or send output to more than one destination simultaneously.

Syntax

```
DEFINE [ [ NEW [ GLOBAL ] ] SHARED ] [ PRIVATE ] STREAM stream
```

`NEW SHARED STREAM` *stream*

Defines and identifies a stream that can be shared by other procedures. When the procedure using the DEFINE NEW SHARED STREAM statement ends, the stream is no longer available to any procedure. The value you use for the *stream* option must be a constant.

`NEW GLOBAL SHARED STREAM` *stream*

Defines and identifies a stream that can be shared by other procedures and that will remain available even after the procedure that contains the DEFINE NEW GLOBAL SHARED STREAM statement ends. The value you use for the *stream* option must be a constant.

`SHARED STREAM` *stream*

Defines and identifies a stream that was created by another procedure using the DEFINE NEW SHARED STREAM statement or the DEFINE NEW GLOBAL SHARED STREAM statement. The value you use for the *stream* option must be a constant.

`[ PRIVATE ] STREAM` *stream*

Defines and identifies a stream as a data member for a class, and optionally specifies an access mode for that data member. Do not specify the access mode when defining a stream for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

STREAM *stream*

Defines and identifies a stream that can be used only by the procedure, or method within a class, containing the DEFINE STREAM statement. The value you use for the *stream* option must be a constant.

Examples

This procedure, in a single pass through the item table, uses the rpt stream to create a report and the exceptions stream to create a list of exceptions:

r-dfstr.p

```
DEFINE NEW SHARED STREAM rpt.
DEFINE STREAM exceptions.
DEFINE VARIABLE fnr AS CHARACTER FORMAT "x(12)".
DEFINE VARIABLE fne AS CHARACTER FORMAT "x(12)".
DEFINE VARIABLE excount AS INTEGER LABEL "Total Number of exceptions".
DEFINE NEW SHARED BUFFER xitem FOR item.

SET fnr LABEL "Enter filename for report output" SKIP(1)
    fne LABEL "Enter filename for exception output"
    WITH SIDE-LABELS FRAME fnames.

OUTPUT STREAM rpt TO VALUE(fnr) PAGED.
OUTPUT STREAM exceptions TO VALUE(fne) PAGED.

FOR EACH xitem:
    IF on-hand < alloc THEN DO:
      DISPLAY STREAM exceptions item-num item-name on-hand alloc
         WITH FRAME exitem DOWN.
      excount = excount + 1.
    END.
    RUN r-dfstr2.p.
END.

DISPLAY STREAM exceptions SKIP(1) excount WITH FRAME exc SIDE-LABELS.
DISPLAY STREAM rpt WITH FRAME exc.

OUTPUT STREAM rpt CLOSE.
OUTPUT STREAM exceptions CLOSE.
```

Include the DISPLAY statement in the `r-dfstr2.p` procedure in the `r-dfstr.p` procedure for efficiency. (It is in a separate procedure here to illustrate shared streams.)

r-dfstr2.p

```
DEFINE SHARED STREAM rpt.
DEFINE SHARED BUFFER xitem FOR item.

DISPLAY STREAM rpt item-num item-name WITH NO-LABELS NO-BOX.
```

Notes

- You cannot define a SHARED or NEW SHARED stream in a user-defined function, an internal procedure, or a persistent procedure. If you do, Progress raises an ERROR on the RUN statement that creates the procedure.
- You cannot define a SHARED or NEW SHARED stream in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- Progress automatically provides two unnamed streams to each procedure: the input stream and the output stream. These streams give the procedure a way to communicate with an input source and an output destination. For example, the following statement tells Progress to use the unnamed input stream to get input from the file named `testfile`.

```
INPUT FROM testfile.
```

- Using the DEFINE STREAM statement creates a stream, but it does not actually open that stream. To open a stream, you must use the STREAM option with the INPUT FROM, INPUT THROUGH, OUTPUT TO, OUTPUT THROUGH, or INPUT-OUTPUT THROUGH statements. You must also use the STREAM option with any data handling statements that move data to and from the stream.
- After you open the stream, you can use the SEEK function to return the offset value of the file pointer, or you can use the SEEK statement to position the file pointer to any location in the file.
- For more information on using streams, see the section on alternate I/O sources in *OpenEdge Development: Programming Interfaces*.

See also

DISPLAY statement, INPUT CLOSE statement, INPUT FROM statement, INPUT THROUGH statement, INPUT-OUTPUT THROUGH statement, OUTPUT CLOSE statement, OUTPUT THROUGH statement, OUTPUT TO statement, PROMPT-FOR statement, RUN statement, SEEK function, SEEK statement, SET statement

DEFINE SUB-MENU statement

Defines a submenu. You can use a submenu as a pull-down menu within a menu bar or as a submenu of a pull-down menu or pop-up menu.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ PRIVATE ] SUB-MENU submenu
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ FGCOLOR expression ]
  [ PFCOLOR expression ]
  [ FONT number ]
  [ SUB-MENU-HELP ]
  { LIKE menu | menu-element-descriptor ... }
```

`[ PRIVATE ] SUB-MENU` *submenu*

Defines and identifies a submenu as a data member for a class, and optionally specifies an access mode for that data member. Do not specify the access mode when defining a submenu for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

`SUB-MENU` *submenu*

Identifies the name of the submenu you are defining. You can define the submenu in a procedure or a method within a class.

`BGCOLOR` *expression*

Specifies the background color for the submenu in graphical interfaces. This option is ignored in character interfaces and Windows.

`DCOLOR` *expression*

Specifies the display color for the submenu in character interfaces. This option is ignored in graphical interfaces.

`FGCOLOR` *expression*

Specifies the foreground color for the submenu in graphical interfaces. This option is ignored in character interfaces.

`PFCOLOR` *expression*

Specifies the prompt-for color for the submenu in character interfaces. This option is ignored in graphical interfaces.

`FONT` *number*

Has no effect; supported only for backward compatibility.

`SUB-MENU-HELP`

Has no effect; supported only for backward compatibility.

`LIKE` *menu*

Specifies a previously defined menu or submenu whose characteristics you want to apply to the new submenu. If you name a menu with this option, you must have previously defined that menu in the procedure. If you name a submenu with this option, that submenu must have already been used as part of a menu definition.

menu-element-descriptor

Specifies an element displayed on the menu. Each element is either a choosable menu item, a submenu, non-choosable text, a rule, or a blank space. You must specify one or more menu elements, unless you use the LIKE option.

This is the syntax for *menu-element-descriptor*:

```
{
   RULE
   SKIP
   SUB-MENU submenu [ DISABLED ] [ LABEL label ]
   menu-item-phrase
}
```

RULE

Specifies that a rule or line is inserted at this point in the submenu. You can use this, for example, to divide the submenu into sections.

SKIP

Specifies that a blank line is inserted at this point in the submenu. You can use this, for example, to divide the submenu into sections.

SUB-MENU *submenu* [DISABLED] [LABEL *label*]

Specifies that a submenu is displayed at this menu item. The submenu must be previously defined in the procedure. The submenu appears when the user chooses that item. The submenu cannot be a menu bar. The DISABLED and LABEL options for a submenu are the same as described for the *menu-item-phrase*.

menu-item-phrase

Specifies a choosable menu item. This is the syntax for *menu-item-phrase*:

```
MENU-ITEM menu-item-name
  [ ACCELERATOR keylabel ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ DISABLED ]
  [ FGCOLOR expression ]
  [ FONT expression ]
  [ LABEL label ]
  [ PFCOLOR expression ]
  [ READ-ONLY ]
  [ TOGGLE-BOX ]
  { [ trigger-phrase ] }
```

MENU-ITEM *menu-item-name*

The name of the menu item you are defining.

ACCELERATOR *keylabel*

Specifies a keyboard accelerator for this menu item. A keyboard accelerator is a key—possibly modified by **SHIFT**, **CONTROL**, or **ALT**—that chooses a menu item even if the menu is not displayed. The value *keylabel* must be character-string expression that evaluates to a valid key label recognized by Progress, such as a, **F1**, or **ALT+SHIFT+F1**.

BGCOLOR *expression*

Specifies the background color for the menu item in graphical interfaces. If you omit this option, the menu item inherits the background color of the submenu.

DCOLOR *expression*

Specifies the display color for the menu item in character interfaces. If you omit this option, the menu item inherits the display color of the submenu.

DISABLED

Specifies that the menu item is initially disabled for input. This means that the user cannot choose this item. Disabled items are grayed out in environments that support it.

FGCOLOR *expression*

Specifies the foreground color for the menu item in graphical interfaces. If you omit this option, the menu item inherits the foreground color of the submenu.

FONT *expression*

Specifies the font for the menu item. If you omit this option, the menu item inherits the font of the menu.

LABEL *label*

Specifies the text that displays in the submenu for a choosable menu item or submenu. If you omit LABEL, Progress displays the item handle by default.

You can include an ampersand (&) within the label to indicate that the following letter acts as a mnemonic for the menu item. This means that when the menu is displayed, the user can choose the item by pressing that single key. If you do not include an ampersand within the label, Windows treats the first character as a mnemonic. To include a literal ampersand within a label, specify a double ampersand (&&).

PFCOLOR *expression*

Specifies the prompt-for color for the menu item in character interfaces. If you omit this option, the menu item inherits the prompt-for color of the submenu.

READ-ONLY

Specifies that this menu item is read-only text. The user cannot choose this item.

TOGGLE-BOX

Specifies that the menu item is displayed as a checkbox that the user can toggle on or off. In environments that do not support this option, it is ignored.

trigger-phrase

Specifies application triggers for the menu item. Typically, you associate a CHOOSE trigger with each menu item.

For more information, see the Trigger phrase reference entry.

Example

The `r-menu.p` procedure defines three pull-down submenus. One of the submenus, myedit, contains a nested submenu, myobjects. The procedure defines a menu bar, mybar, that contains two submenus labelled File and Edit. The handle of mybar is assigned to a window mywin. The ON statements define triggers to execute when you choose the corresponding menu items.

r-menu.p

```
DEFINE VARIABLE mywin AS WIDGET-HANDLE.
DEFINE SUB-MENU myfile
   MENU-ITEM m1  LABEL "Save"
   MENU-ITEM m2  LABEL "Save As"
   MENU-ITEM m3  LABEL "Exit".DEFINE SUB-MENU myobjects
   MENU-ITEM m1  LABEL "Circle"
   MENU-ITEM m2  LABEL "Line"
   MENU-ITEM m3  LABEL "Rectangle"
   MENU-ITEM m4  LABEL "Text".DEFINE SUB-MENU myedit
   SUB-MENU myobjects LABEL "Add"
   MENU-ITEM e1       LABEL "Delete"
   MENU-ITEM e2       LABEL "Copy".DEFINE MENU mybar  MENUBAR
   SUB-MENU myfile LABEL "File"
   SUB-MENU myedit LABEL "Edit".
CREATE WINDOW mywin
   ASSIGN MENUBAR = MENU mybar:HANDLE.DEFINE BUTTON b1 LABEL "Text Mode".
DEFINE BUTTON b2 LABEL "Graphics Mode".CURRENT-WINDOW = mywin.
FORM
   b1 at X 10  Y 120
   b2 at x 120 Y 120
   WITH FRAME x.ENABLE b1 b2 WITH FRAME x.
ON CHOOSE OF b1 IN FRAME x DO:
   MENU-ITEM m1:SENSITIVE IN MENU myobjects = NO.
   MENU-ITEM m2:SENSITIVE IN MENU myobjects = NO.
   MENU-ITEM m3:SENSITIVE IN MENU myobjects = NO.
   MENU-ITEM m4:SENSITIVE IN MENU myobjects = YES.
END.

ON CHOOSE OF b2 IN FRAME x DO:
   MENU-ITEM m1:SENSITIVE IN MENU myobjects = YES.
   MENU-ITEM m2:SENSITIVE IN MENU myobjects = YES.
   MENU-ITEM m3:SENSITIVE IN MENU myobjects = YES.
   MENU-ITEM m4:SENSITIVE IN MENU myobjects = NO.
END.

WAIT-FOR CHOOSE OF MENU-ITEM m3 IN MENU myfile.
DELETE WIDGET mywin.
```

Notes

- To create the static submenu you are defining, along with any of its descendents (submenus and menu items), you must define a static menu that contains the submenu. Each menu you define that contains the same submenu creates an additional instance of the submenu and each of its descendents. The widget handles for a static submenu and its descendents are not available until the submenu is created in a menu.
- You cannot define a submenu with the same name more than once in the same menu tree. Thus, if menu mFile contains both submenu mOptions and submenu mSave, submenu mSave cannot also contain submenu mOptions.
- Menu items in different menus and submenus can have the same names. In the above procedure, the menu items in myfile and myobjects share the same names. To avoid ambiguity, use the IN MENU or IN SUB-MENU option to identify the parent menu or submenu.
- There are instances where you cannot avoid ambiguity in menu item references. In such instances, Progress always references the first unambiguous instance of the menu item. In particular, if the same submenu containing a menu item appears in more than one menu and each menu defines another instance of the same menu item, you can only reference that menu item in the submenu from the **first** menu that contains it. Thus, if submenu mOptions contains menu item mSave and the menus mFile and mDraw (in that order) both contain submenu mOptions and another menu item mSave, you can only reference menu item mSave in submenu mOptions from menu mFile. You cannot uniquely reference menu item mSave in submenu mOptions from menu mDraw because menu mDraw contains another menu item mSave. For more information on menu item references, see the chapter on menus in *OpenEdge Development: Progress 4GL Handbook*.
- When a menu item is disabled, it appears grayed-out (if the environment supports that) and it cannot be chosen.

See also CREATE widget statement, Trigger phrase

DEFINE TEMP-TABLE statement

Defines a temporary table. Progress stores temporary tables on disk in a temporary database. A temporary table can be either global (lasting for the entire OpenEdge session) or local (lasting only as long as the procedure that creates it), and either shared (visible to other procedures that want to access it) or non-shared (visible just to the procedure that created it).

Syntax

```
DEFINE [ [ NEW [ GLOBAL ] ] SHARED ]
  [ PRIVATE | PROTECTED ] TEMP-TABLE temp-table-name [ NO-UNDO ]
  [ NAMESPACE-URI namespace ] [ NAMESPACE-PREFIX prefix ]
  [ REFERENCE-ONLY ]
  [ LIKE table-name
      [ VALIDATE ]
      [ USE-INDEX index-name [ AS PRIMARY ] ] ... ]
  [ RCODE-INFORMATION ]
  [ BEFORE-TABLE before-table-name ]
  [ FIELD field-name
      { AS data-type |  LIKE field [ VALIDATE ] }
  [ field-options ]
  ] ...
  [ INDEX index-name
      [ IS [ UNIQUE ] [ PRIMARY ] [ WORD-INDEX ] ]
      { index-field [ ASCENDING | DESCENDING ] } ...
  ] ...
```

NEW SHARED TEMP-TABLE *temp-table-name*

Defines and identifies a temporary table that can be shared by one or more procedures called directly or indirectly by the current procedure. The temporary table remains available to other procedures until the procedure that defined it ends. The called procedures must define the same temporary table name as SHARED.

SHARED TEMP-TABLE *temp-table-name*

Defines and identifies a temporary table that was initially defined by another procedure as NEW SHARED.

The procedure that establishes the temporary table determines the name. The procedures that share the temporary table use that name to identify it.

NEW GLOBAL SHARED TEMP-TABLE *temp-table-name*

Defines and identifies a global shared temporary table, and accesses an existing one. The scope of a global shared temporary table is the OpenEdge session. The first procedure to define a temporary table NEW GLOBAL SHARED establishes it. Subsequent procedures access it.

Note: Progress does not establish multiple global shared temporary tables with the same name in the same OpenEdge session.

[PRIVATE | PROTECTED] TEMP-TABLE *temp-table-name*

Defines and identifies a temporary table as a data member for a class, and optionally specifies an access mode for that data member. Do not specify an access mode when defining a temporary table for a method within a class.

PRIVATE data members can be accessed only by the defining class. PROTECTED data members can be accessed by the defining class and any of its inheriting classes. The default access mode is PRIVATE.

Note: These options are applicable only when defining a data member for a class in a class definition (`.cls`) file.

TEMP-TABLE *temp-table-name*

Identifies the name of the temporary table. You can define the temporary table in a procedure, a method within a class, or an interface.

NO-UNDO

Specifies that when a transaction is undone, changes to the temporary table records need not be undone. If you do not specify this option, all records in the temporary table are restored to their prior condition when a transaction is undone. The NO-UNDO option can significantly increase the performance for temporary table updates; use it whenever possible.

NAMESPACE-URI *namespace*

An optional CHARACTER constant that specifies the URI for the namespace of the temp-table.

NAMESPACE-PREFIX *prefix*

An optional CHARACTER constant that specifies the namespace prefix associated with the NAMESPACE-URI.

REFERENCE-ONLY

Specifies that the procedure defining this temporary table object is using the object definition only as a reference to a temporary table object that is defined and instantiated in another procedure or class, and specified as a parameter in the invocation of a RUN statement, a method in a class, or a user-defined function, using either the BY-REFERENCE or BIND option. Progress does not instantiate the reference-only object.

Passing a reference-only temporary table object parameter to a local routine using either the BY-REFERENCE or BIND option allows the calling routine and the called routine to access the same object instance (instead of deep-copying the parameter).

Note: If you pass the parameter to a remote procedure, Progress deep-copies the parameter on OUTPUT and the reference-only parameter is bound to that copy.

When you pass a temporary table parameter to a local routine using the BY-REFERENCE option, both the calling and called routines access the calling routine's object instance (and ignore the called routine's object instance). Since the called routine's object instance is ignored, you should define the object as a reference-only object. When you define a reference-only temporary table object in the called routine and receive it from the calling routine using the BY-REFERENCE option, Progress binds the definition of the object in the called routine to the object instance in the calling routine for the duration of the called routine. You cannot define a reference-only temporary table object in the calling routine and pass it to the called routine using the BY-REFERENCE option.

When you pass a temporary table parameter to a local routine using the BIND option, you can define a reference-only temporary table object in either the calling routine or the called routine as follows:

- When you define a reference-only temporary table object in the calling routine and pass it to the called routine using the BIND option, Progress binds the calling routine to the object instance in the called routine. The reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates. The parameter must be an OUTPUT parameter.

  Note: If you also define the temporary table object instance in the called routine as a reference-only object, you must bind the object in the called routine before returning to the calling routine.

- When you define a reference-only temporary table object in the called routine and receive it from the calling routine using the BIND option, Progress binds the called routine to the object instance in the calling routine. The reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates. The parameter must be an INPUT or INPUT-OUTPUT parameter.

In either case, you must specify the BIND option for the parameter in both the invocation of a RUN statement, a method in a class, or a user-defined function, and in the DEFINE PARAMETER statement.

Caution: Do not delete the object or routine to which a reference-only temporary table object is bound, or you might be left with references to an object that no longer exists.

A reference-only temporary table object can be a member of a reference-only ProDataSet object or a standard ProDataSet object. However, if you define a reference-only temporary table in a standard ProDataSet object, you cannot use the ProDataSet object until you bind the reference-only temporary table.

```
LIKE table-name [ USE-INDEX index-name  [ AS PRIMARY ] ] . . .
```

Specifies the name of a table whose characteristics the temporary table inherits. All field definitions of *table-name* are added to the temporary table. *table-name* can represent a database table or another temporary table.

If you reference a database field, the database containing that field must be connected at compile time. If the database field has a validation expression defined in the dictionary that contains a database reference, and the VALIDATE option is specified, the database must also be connected at runtime.

HELP options are inherited from the *table-name*. Validate options are inherited only if the VALIDATE keyword is used.

Some index definitions from the specified table might also be added to the temporary table:

- If you use the USE-INDEX option, only the definitions of indexes you specify with that option are copied to the temporary table. If one of these indexes is the primary index of the LIKE table, it becomes the default primary index of the temporary table. You can, however, use the AS PRIMARY option to override this default primary index.

  For example, to make the index country-post the primary index (thereby, overriding the default primary index cust-num in the table customer), you specify it as follows:

  ```
  DEFINE TEMP-TABLE mycust LIKE customer
    USE INDEX cust-num USE INDEX country-post AS PRIMARY.
  ```

- If you do not specify the USE-INDEX option and do not use the INDEX option of the DEFINE TEMP-TABLE statement, then all index definitions are copied from the specified table to the temporary table. In this case, the primary index of the specified table becomes the primary index of the temporary table.
- If you do not specify the USE-INDEX option but do use the INDEX option of the DEFINE TEMP-TABLE statement, then no indexes are copied from the specified table.
- Progress does not copy inactive indexes to the temporary table.

VALIDATE

The temp-table fields inherit, from the dictionary, validation expressions and validation messages from the database table, *table-name*.

RCODE-INFORMATION

Note: This option is supported only for backward compatibility.

BEFORE-TABLE *before-table-name*

Specifies the name of the before-image table associated with a static temp-table in a ProDataSet object. You must specify a before-image table name for any static ProDataSet temp-table for which you want to track changes. If you try to modify the records in this before-image table, Progress generates a runtime error.

FIELD *field-name*

Defines a field in the temporary table. You can use FIELD clauses with the LIKE option to define additional fields for the temporary table, or you can define all your fields with FIELD clauses.

AS *data-type*

Specifies the data type of the field. The valid data types are BLOB, CHARACTER, CLASS, CLOB, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INTEGER, LOGICAL, RAW, RECID, ROWID and WIDGET-HANDLE.

For the CLASS data type, you define a field in a temporary table as a class by specifying the built-in Progress.Lang.Object class name. For example:

```
DEFINE TEMP-TABLE ttObjHolder FIELD MyObj AS CLASS Progress.Lang.Object.
```

When you assign a class object instance to a field, Progress implicitly casts the instance to its root super class, which is the Progress.Lang.Object class. After the assignment, the field contains an object reference for a class object instance, not the class itself.

You cannot define a field in a database table as a class.

Note: When a temporary table contains one or more fields defined with the Progress.Lang.Object class, you cannot pass the temporary table to an AppServer.

`LIKE` *`field`*

Specifies a database field or a variable whose characteristics the temporary table field inherits. If you name a variable with this option, that variable must have been defined earlier in the procedure. The temporary table field inherits the data type, extents, format, initial value, label, and column label.

If the database field is a `COLUMN-CODEPAGE` CLOB, the temp-table field is in the database field's code page. If the database field is a `DBCODEPAGE` CLOB, the temp-table field's code page is `-cpinternal`.

You can override selected characteristics of the field or variable with the *`field-options`* parameter.

If you reference a database field in the LIKE option, the database containing that field must be connected at both compile time and run time. Therefore, use the LIKE option with caution.

field-options

Specifies options for the temporary table field. Any options you specify override any options inherited through the LIKE option. This is the syntax for *field-options*:

```
{
   [ BGCOLOR expression ]
   [ COLUMN-LABEL label ]
   [ DCOLOR expression ]
   [ DECIMALS n ]
   [ EXTENT n ]
   [ FONT expression ]
   [ FGCOLOR expression ]
   [ FORMAT string ]
   [ HELP help-text ]
   [ INITIAL
      { constant | { [ constant [ , constant ] ... ] } }
  ]
   [ LABEL label [ , label ] ... ]
   [ MOUSE-POINTER expression ]
   [ [ NOT ] CASE-SENSITIVE ]
   [ PFCOLOR expression ]
   [ TTCODEPAGE | COLUMN-CODEPAGE codepage ]
   [ XML-DATA-TYPE string ]
   [ XML-NODE-TYPE string ]
  { [ view-as-phrase ] }
 }
```

HELP

A quoted CHARACTER string that represents the help text.

TTCODEPAGE | COLUMN-CODEPAGE *codepage*

Specifies the code page for a CLOB field in the temporary table. If you specify TTCODEPAGE, the code page is -cpinternal. If you specify COLUMN-CODEPAGE, *codepage* must be a valid code page name available in the DLC/convmap.cp file. You cannot specify the "undefined" code page for a CLOB. The code page you specify overrides any code page inherited through the LIKE option.

If you do not specify a code page for a CLOB field in the temporary table, the default code page is -cpinternal.

XML-DATA-TYPE *string*

An optional CHARACTER constant that specifies the XML Schema data type for the field in the temporary table. The XML Schema data type must be compatible with the Progress data type for the field.

For more information about the Progress XML data type mapping rules, see *OpenEdge Development: Programming Interfaces*.

XML-NODE-TYPE *string*

An optional CHARACTER constant that specifies the XML node type of the temp-table field, which lets you specify how the field is represented in XML. Valid XML node types are: "ATTRIBUTE", "ELEMENT", "HIDDEN", and "TEXT". The default value is "ELEMENT".

Table 28 lists the valid XML node types.

Table 28: XML node types *(1 of 2)*

When the XML node type is ...	The buffer field is ...
ATTRIBUTE	Represented as an attribute of the temp-table element in both the XML Schema and data.
ELEMENT	Represented as a child element of the temp-table element in both the XML schema and data.

Table 28: XML node types *(2 of 2)*

When the XML node type is ...	The buffer field is ...
HIDDEN	Omitted from both the XML Schema and data.
TEXT	Represented as a text element in both the XML Schema and data. **Note:** Each table can contain only one TEXT field. When a table contains a TEXT field, it cannot contain ELEMENT fields; it can contain only ATTRIBUTE fields. A table that contains a TEXT field cannot be part of a nested data-relation.

The XML node type of a temp-table field that represents an array must be either "ELEMENT" or "HIDDEN".

Note: You cannot specify an indeterminate array field in a temp-table using the EXTENT field option.

For more information about the EXTENT field option, and a description of the other field options, see the DEFINE VARIABLE statement.

```
INDEX index-name [ IS [ UNIQUE ] [ PRIMARY ] [ WORD-INDEX ] ]
```

Defines an index on the temporary table. To define a unique index, specify the UNIQUE option. To define the primary index, specify the PRIMARY option. To define a word-index, specify the WORD-INDEX option.

If you define more that one index on the temporary table, you can specify PRIMARY for none or one of the indexes. If you specify PRIMARY for none of the indexes, Progress makes the first index you specify the primary index.

If you define no indexes on the temporary table, and the temporary table does not inherit the indexes of another table through the LIKE option of the DEFINE TEMP-TABLE statement, Progress creates a default index, makes it the primary index, and sorts the records in entry order.

index-field [ASCENDING | DESCENDING]

Specifies a temporary table field to use as a component of the index. You can use the ASCENDING or DESCENDING option to specify that the component has ascending or descending order.

If you do not specify a sort orientation (ASCENDING or DESCENDING), the index component gets the sort orientation of the previous index component, or, if there is no previous index component, ASCENDING. This rule applies only to index components of temp-tables.

Note: You cannot use a BLOB or CLOB field as a component of an index.

For example, the following two temp-table definitions are equivalent:

```
DEFINE TEMP-TABLE foo FIELD a AS CHAR FIELD b AS CHAR FIELD c AS CHAR
  INDEX x a DESC b DESC c DESC.
```

```
DEFINE TEMP-TABLE foo FIELD a AS CHAR FIELD b AS CHAR FIELD c AS CHAR
  INDEX x a DESC b c.
```

The following two temp-table definitions are also equivalent:

```
DEFINE TEMP-TABLE foo FIELD a AS CHAR FIELD b AS CHAR FIELD c AS CHAR
  INDEX x a ASC b DESC c DESC.
```

```
DEFINE TEMP-TABLE foo FIELD a AS CHAR FIELD b AS CHAR FIELD c AS CHAR
  INDEX x a ASC b DESC c.
```

Examples

The following procedure creates a temporary table (`tempitem`) that stores the total inventory value (item.price * item.on-hand) for each catalog page (item.cat-page) in the sports database. It builds temp-item with two indexes-one that sorts the table in ascending order by catalog page and a second that sorts the table in descending order by inventory value.

After building temp-item, the procedure displays a dialog box that prompts for report parameters. These parameters include the cutoff value of catalog page inventory to report, and whether to display the report by catalog page (ascending) or inventory value (descending). After displaying the report, the procedure displays another dialog box to repeat the process. The process is repeated until you press the CANCEL button. This procedure shows how you can use a temporary table to store a calculated result from the database, and efficiently report the same result according to different sorting and selection criteria:

r-tmptb1.p

```
DEFINE TEMP-TABLE temp-item
   FIELD cat-page LIKE item.cat-page
   FIELD inventory LIKE item.price LABEL "Inventory Value"
   INDEX cat-page IS PRIMARY cat-page ASCENDING
   INDEX inventory-value inventory DESCENDING.
DEFINE VARIABLE cutoff LIKE item.price.
DEFINE VARIABLE inv-value LIKE item.price.
DEFINE VARIABLE report-type AS INTEGER INITIAL 1.
DEFINE BUTTON ok-butt LABEL "OK" AUTO-GO.
DEFINE BUTTON cancel-butt LABEL "CANCEL" AUTO-ENDKEY.
FORM
   cutoff LABEL "Inventory Lower Cutoff for each Catalog Page"
      AT ROW 1.25 COLUMN 2
   report-type LABEL "Report Sorted ..."
      AT ROW 2.25 COLUMN 2
      VIEW-AS RADIO-SET RADIO-BUTTONS
         "By Catalog Page",   1,
         "By Inventory Value", 2
   SKIP ok-butt cancel-butt
   WITH FRAME select-frame SIDE-LABELS WIDTH 70
      TITLE "Specify Report ..." VIEW-AS DIALOG-BOX.
FOR EACH item BREAK BY item.cat-page:
   ACCUMULATE price * on-hand (SUB-TOTAL BY item.cat-page).
   IF LAST-OF(item.cat-page) THEN DO:
      inv-value = ACCUM SUB-TOTAL BY item.cat-page (price * on-hand).
      CREATE temp-item.
      temp-item.cat-page = item.cat-page.
      inventory = inv-value.
   END.
END. /* FOR EACH item */
ON CHOOSE OF ok-butt
  DO:
    HIDE FRAME select-frame.
    IF report-type = 1 THEN
       FOR EACH temp-item USE-INDEX cat-page WITH FRAME rpt1-frame:
          IF inventory >= cutoff THEN
             DISPLAY temp-item.cat-page inventory.
       END.
    ELSE
       FOR EACH temp-item USE-INDEX inventory-value WITH FRAME rpt2-frame:
          IF inventory >= cutoff THEN
             DISPLAY temp-item.cat-page inventory.
       END.
    VIEW FRAME select-frame.
  END.
ENABLE ALL WITH FRAME select-frame.
WAIT-FOR CHOOSE OF cancel-butt OR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- If you define a temporary table LIKE a database table, the temporary table does not inherit the database table's database triggers.
- You cannot define a temporary table field of type MEMPTR or LONGCHAR.
- You cannot define shared objects, work tables, or temporary tables within an internal procedure, a method in a class, or a user-defined function.
- Progress disregards the following when used in conjunction with a temporary table:
  - The VALIDATE option on a DELETE statement.
  - The SHARE-LOCK, EXCLUSIVE-LOCK, and NO-LOCK options used with the FIND or FOR statements.
  - The NO-WAIT option on the FIND statement.
- Data handling statements that cause Progress to automatically start a transaction for a regular table will not cause Progress to automatically start a transaction for a temporary table. If you want to start a transaction for operations involving a temporary table, you must explicitly start a transaction by using the TRANSACTION keyword.
- Use the CASE-SENSITIVE option only when it is important to distinguish between uppercase and lowercase values entered for a character field. For example, use CASE SENSITIVE to define a field for a part number that contains mixed upper-case and lowercase characters.
- You cannot define a SHARED or NEW SHARED temporary table in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- A SHARED temporary table remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the temporary table as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED temporary table, Progress includes the persistent procedure in the resolution of the corresponding NEW SHARED temporary table as though the procedure were on the procedure call stack.

- You can specify a join between a temporary table or work table and any appropriate table using the OF keyword. The two tables must contain a commonly named field that participates in a unique index for at least one of the tables. For more information on table joins see the Record phrase reference entry.
- If you define a temporary table with the same name as a database table and then you define a buffer for that name, the buffer will be associated with the database table, not with the temporary table.
- See *OpenEdge Development: Progress 4GL Handbook* for information on temporary tables and work tables.

See also CREATE TEMP-TABLE statement, DEFINE DATASET statement, DEFINE WORK-TABLE statement, NUM-REFERENCES attribute, RUN statement

DEFINE VARIABLE statement

Defines a variable for use within one or more procedures, or defines a variable as a data member within a class.

Syntax

```
DEFINE
  { [ [ NEW [ GLOBAL ] ] SHARED ] | [ PRIVATE | PROTECTED | PUBLIC ] }
  VARIABLE variable-name
  {  AS datatype
   | AS [ CLASS ] { type-name }
   | LIKE field }
  [ BGCOLOR expression ]
  [ COLUMN-LABEL label ]
  [ CONTEXT-HELP-ID expression ]
  [ DCOLOR expression ]
  [ DECIMALS n ]
  [ DROP-TARGET ]
  [ EXTENT [ expression ] ]
  [ FONT expression ]
  [ FGCOLOR expression ]
  [ FORMAT string ]
  [ INITIAL
     { constant | { [ constant [ , constant ] ... ] } } ]
  [ LABEL string [ , string ] ... ]
  [ MOUSE-POINTER expression ]
  [ NO-UNDO ]
  [ [ NOT ] CASE-SENSITIVE ]
  [ PFCOLOR expression ]
  { [ view-as-phrase ] }
  { [ trigger-phrase ] }
```

`NEW SHARED VARIABLE variable-name`

Defines and identifies a variable to be shared by a procedure called directly or indirectly by the current procedure. The called procedure must name the same variable in a DEFINE SHARED VARIABLE statement.

`NEW GLOBAL SHARED VARIABLE` *variable-name*

Defines and identifies a variable that can be used by any procedure that names that variable using the DEFINE SHARED VARIABLE statement. The value of a global shared variable remains available throughout an OpenEdge session.

`SHARED VARIABLE` *variable-name*

Defines and identifies a variable that was created by another procedure that used the DEFINE NEW SHARED VARIABLE or DEFINE NEW GLOBAL SHARED VARIABLE statement.

[`PRIVATE` **|** `PROTECTED` **|** `PUBLIC` **]** `VARIABLE` *variable-name*

Defines and identifies a variable as a data member for a class, and optionally specifies an access mode for that data member. Do not specify an access mode when defining a variable for a method within a class.

The variable data member name must be unique among all data members in the defining class and its inherited class hierarchy.

PRIVATE data members can be accessed only by the defining class. PROTECTED data members can be accessed by the defining class and any of its inheriting classes. PUBLIC data members can be accessed by the defining class, any of its inheriting classes, and any class or procedure that instantiates that class. The default access mode is PRIVATE.

Note: These options are applicable only when defining a data member for a class in a class definition (`.cls`) file.

`VARIABLE` *variable-name*

Defines and identifies a variable whose value is available only within the current procedure, or a method within a class.

`AS` *datatype*

Indicates the data type of the variable you are defining. The data types are CHARACTER, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INTEGER, LONGCHAR, LOGICAL, MEMPTR, RAW, RECID, ROWID, and WIDGET-HANDLE.

AS [CLASS] { *type-name* }

Defines an object reference variable for the specified class or interface. The default value of the variable is the Unknown value (?).

You use this object reference variable with the NEW statement to create a new instance of a class. You access a class object instance, as well as its data members and methods, using this object reference variable. For more information about the NEW statement, see the NEW statement reference entry in this book.

Note: You can define an object reference variable for an interface, which lets you access the interface or a class that implements the interface, but you **cannot** create an instance of an interface with the NEW statement.

type-name

A character string that specifies the type name of the class, a subclass of the class, or the interface for which this object reference variable is defined. Specify a type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

If the specified class or interface type name conflicts with an abbreviation of a built-in Progress data type name, such as INT for INTEGER, you must specify the CLASS keyword.

LIKE *field*

Indicates the name of the variable, database field, temporary table field, or work table field whose characteristics you want to use for the variable you are defining. If you name a variable with this option, you must have defined that variable earlier in the procedure. You can override the format, label, initial value, decimals, and extent of the variable or database field by using the FORMAT, LABEL, COLUMN-LABEL, INITIAL, DECIMALS, EXTENT, and VIEW-AS options. If you do not use these options, the variable takes on the characteristics of the variable or database field you name.

If *field* has help and validate options defined, the variable you are defining does **not** inherit those characteristics.

If you reference a database field in a LIKE option in a DEFINE VARIABLE statement, DEFINE TEMP-TABLE statement, DEFINE WORK-TABLE statement, or format phrase, the database containing the referenced field must be connected at both compile time and runtime. Therefore, use the LIKE option with caution.

BGCOLOR *expression*

Specifies a background color for the variable in graphical interfaces. This option is ignored in character interfaces.

[NOT] CASE-SENSITIVE

CASE-SENSITIVE indicates that the value stored for a character variable is case sensitive, and that all comparisons operations involving the variable are case sensitive. If you do not use this option, Progress comparisons are usually case insensitive. If you define a variable LIKE another field of variable, the new variable inherits case sensitivity. Use [NOT] CASE-SENSITIVE to override this default.

COLUMN-LABEL *label*

Names the label you want to display above the variable data in a frame that uses column labels. If you want the label to use more than one line (a stacked label), use an exclamation point (!) in the label to indicate where to break the line.

r-collbl.p

```
DEFINE VARIABLE credit-percent AS INTEGER
    COLUMN-LABEL "Enter   !percentage!increase ".

FOR EACH customer:
    DISPLAY name credit-limit.
    SET credit-percent.
    credit-limit = (credit-limit * (credit-percent / 100))
                   + credit-limit.
    DISPLAY credit-limit @ new-credit LIKE credit-limit
        LABEL "New max cred".
END.
```

If you want to use the exclamation point (!) as one of the characters in a column label, use two exclamation points (!!).

Progress does not display column labels if you use the SIDE-LABELS or NO-LABELS options with the Frame phrase.

If you define a variable to be LIKE a field, and that field has a column label in the Data Dictionary, the variable inherits that column label.

CONTEXT-HELP-ID *expression*

An integer value that specifies the identifier of the help topic for this variable in a help file specified at the session, window or dialog box level using the CONTEXT-HELP-FILE attribute.

DCOLOR *expression*

Specifies the display color for the variable in character interfaces. This option is ignored in graphical interfaces.

DECIMALS *n*

Specifies the number of decimal places to store for a DECIMAL variable, where *n* is an integer constant. When you define a variable AS DECIMAL, Progress automatically stores up to 10 decimal places for the value of that variable. Use the DECIMALS option to store a smaller number of decimal places. The DECIMALS option has nothing to do with the display format of the variable, just the storage format.

If you use the LIKE option to name a field whose definition you want to use to define a variable, Progress uses the number of decimals in the field definition to determine how many decimal places to store for the variable.

DROP-TARGET

Indicates whether you want to be able to drop a file onto the object.

The following example shows setting the DROP-TARGET option for a variable:

```
DEFINE VARIABLE fill-in-1 AS CHARACTER DROP-TARGET.
```

EXTENT **[** *expression* **]**

Specifies a determinate array variable (which has a defined number of elements) or an indeterminate array variable (which has an undefined number of elements). To define a determinate array variable, specify the EXTENT option with the *expression* argument. This optional argument evaluates to an integer value that represents an extent for the array variable. To define an indeterminate array variable, specify the EXTENT option without the *expression* argument.

If you want to define a variable that is like an array variable or field, using the LIKE option, but you do not want the variable to be an array, you can use EXTENT 0 to indicate a non-array field.

If you are using the AS *datatype* option and you do not use the EXTENT option (or you specify *expression* as 0), the variable is not an array variable. If you are using the LIKE *field* option and you do not use the EXTENT option, the variable uses the extent defined for the database field you name (if any).

Note: You cannot define an array variable as a PUBLIC data member of a class.

FGCOLOR *expression*

Specifies a foreground color for the variable in graphical interfaces. This option is ignored in character interfaces.

FONT *expression*

Specifies a font for the variable.

FORMAT *string*

The data format of the variable you define. If you use the AS *datatype* option and you do not use FORMAT *string*, the variable uses the default format for its data type. Table 29 lists the default data formats for the data types.

Table 29: Default display formats *(1 of 2)*

Data type	Default display format
BLOB[1]	See the footnote at the end of this table
CHARACTER	x(8)
CLASS[3]	>>>>>>9
CLOB[1]	See the footnote at the end of this table
COM-HANDLE	>>>>>>9
DATE	99/99/99
DATETIME	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	->>,>>9.99

Table 29: Default display formats *(2 of 2)*

Data type	Default display format
HANDLE[2]	>>>>>>9
INTEGER	->,>>>,>>9
LOGICAL	yes/no
LONGCHAR[1]	See the footnote at the end of this table
MEMPTR[1]	See the footnote at the end of this table
RAW[1]	See the footnote at the end of this table
RECID	>>>>>>9
ROWID[1]	See the footnote at the end of this table
WIDGET-HANDLE[2]	>>>>>>9

[1] You cannot display a BLOB, CLOB, MEMPTR, RAW, or ROWID value directly. However, you can convert a MEMPTR, RAW, or ROWID value to a character string representation using the STRING function and display the result. You can also convert a BLOB to a MEMPTR, and then use the STRING function. A MEMPTR or RAW value converts to decimal integer string. A ROWID value converts to a hexadecimal string, "0x*hexdigits*," where *hexdigits* is any number of characters "0" through "9" and "A" through "F". You can display a CLOB field by converting it to a LONGCHAR, and displaying the LONGCHAR using the VIEW-AS EDITOR LARGE phrase only.

[2] To display a HANDLE or WIDGET-HANDLE, you must first convert it using the INTEGER function and display the result.

[3] To display a CLASS, you must first convert it using the INTEGER or STRING function and display the result.

See *OpenEdge Development: Progress 4GL Handbook* for more information on data formatting.

If you use the LIKE *field* option and you do not use the FORMAT *string* option, the variable uses the format defined for the database field you name. You must enclose the *string* in quotes.

```
INITIAL { constant | [ constant [ , constant ] . . . ] }
```

The initial value of the variable you want to define. If you use the AS *datatype* option and you do not use the INITIAL *constant* option, the default is the initial value for the data type of the variable.

When you define an array variable, you can supply initial values for each element in the array. For example:

```
DEFINE VARIABLE array-var
  AS CHARACTER EXTENT 3 INITIAL ["Add","Delete","Update"].
```

If you do not supply enough values to fill up the elements of the array, Progress puts the last value you named into the remaining elements of the array. If you supply too many values, Progress returns an error message.

If you define a variable as an indeterminate array, and you supply initial values for elements in the array, Progress fixes the number of elements in the array and treats the fixed indeterminate array as a determinate array. For example, the arrays defined by the following statements are equivalent:

```
DEFINE VARIABLE x AS INT EXTENT INIT [1,2,3].
DEFINE VARIABLE x AS INT EXTENT 3 INIT [1,2,3]
```

Table 30 lists the default initial values for the various variable data types.

Table 30: Default variable initial values *(1 of 2)*

Data type	Default initial value
CHARACTER	"" (an empty string)
CLASS	Unknown value (?)
COM-HANDLE	Unknown value (?)
DATE	Unknown value (?) (displays as blanks)
DATETIME	Unknown value (?)

Table 30: Default variable initial values *(2 of 2)*

Data type	Default initial value
DATETIME-TZ	Unknown value (?)
DECIMAL	0
HANDLE	Unknown value (?)
INTEGER	0
LOGICAL	no
LONGCHAR	Unknown value (?)
MEMPTR	A zero-length sequence of bytes
RAW	A zero-length sequence of bytes
RECID	Unknown value (?)
ROWID	Unknown value (?)
WIDGET-HANDLE	Unknown value (?)

If you are using the LIKE *field* option and you do not use the INITIAL *constant* option, the variable uses the initial value of the field or variable. In the DEFINE SHARED VARIABLE statement, the INITIAL option has no effect. However, the DEFINE NEW SHARED VARIABLE, the DEFINE NEW SHARED TEMP-TABLE, and the DEFINE NEW WORK-TABLE statements work with the INITIAL option.

LABEL *string* **[** , *string* **]** **...**

The label you want to use when the variable is displayed. If you use the AS *datatype* option and you do not use the LABEL *string* option, the default label is the variable name. If you use the LIKE *field* option and you do not use the LABEL *string* option, the variable uses the label of the field or variable you name. You must enclose the *string* in quotes.

You can specify a label for each element in a determinate array variable. You cannot specify a label for elements in an indeterminate array variable.

In MS-Windows, you can designate a character within each label as a navigation mnemonic. Precede the character with an ampersand (&). When the variable is displayed with side labels, the mnemonic is underlined. The user can move focus to the variable by pressing **ALT** and the underlined letter. Navigation mnemonics operate only when you use side labels. If you specify more than one widget with the same mnemonic, Progress transfers focus to each of these in tab order when you make a selection.

Ending a label with an ampersand might produce unwanted behavior. To include a literal ampersand within a label, specify a double ampersand (&&).

`MOUSE-POINTER` *`expression`*

Specifies the default mouse pointer for the variable.

`NO-UNDO`

When the value of a variable is changed during a transaction and the transaction is undone, Progress restores the value of the variable to its prior value. If you do not want, or if you do not need, the value of a variable to be undone even when it has been changed during a transaction, use the NO-UNDO option with the DEFINE VARIABLE statement. NO-UNDO variables are efficient; use this option whenever possible.

Specifying NO-UNDO for a variable is especially useful if you want to indicate an error condition as the value of the variable, perform an UNDO, and later take some action based on that error condition. If one variable is defined LIKE another that is NO-UNDO, the second variable will be NO-UNDO only if you specify NO-UNDO in the definition of the second variable.

`PFCOLOR` *`expression`*

Specifies the prompt-for color for the variable in character interfaces. This option is ignored in graphical interfaces.

view-as-phrase

Specifies the default data representation widget for this variable. Following is the syntax for the *view-as-phrase*:

```
VIEW-AS
  {     combo-box-phrase
      | editor-phrase
      | FILL-IN
          [ NATIVE ]
          [ size-phrase ]
          [ TOOLTIP tooltip ]
      | radio-set-phrase
      | selection-list-phrase
      | slider-phrase
      | TEXT
          [ size-phrase ]
          [ TOOLTIP tooltip ]
      | TOGGLE-BOX
          [ size-phrase ]
          [ TOOLTIP tooltip ]
  }
```

For more information on *view-as-phrase*, see the VIEW-AS phrase reference entry.

trigger-phrase

Defines triggers for the data representation widget specified in the *view-as-phrase*. Following is the syntax for the *trigger-phrase*:

```
TRIGGERS:
  { ON event-list [ ANYWHERE ]
      {     trigger-block
          | PERSISTENT RUN proc-name
              [ IN handle ]
              [ ( input-parameters ) ]
      }
  } ...
END [ TRIGGERS ]
```

For more information on triggers, see the Trigger phrase reference entry.

Examples

The r-dfvar.p procedure defines two variables, del and nrecs to be shared with procedure r-dfvar2.p. The del variable passes information to r-dfvar2.p, while nrecs passes information back to r-dfvar.p from r-dfvar2.p.

r-dfvar.p

```
DEFINE NEW SHARED VARIABLE del AS LOGICAL.
DEFINE NEW SHARED VARIABLE nrecs AS INTEGER.
del = no.
MESSAGE "Do you want to delete the orders
  being printed (y/n)?"
UPDATE del.
RUN r-dfvar2.p.
IF del
THEN MESSAGE nrecs
  "Orders have been shipped and were deleted".
ELSE MESSAGE nrecs "Orders have been shipped".
```

r-dfvar2.p

```
DEFINE SHARED VARIABLE del AS LOGICAL.
DEFINE SHARED VARIABLE nrecs AS INTEGER.

OUTPUT TO PRINTER.
nrecs = 0.
FOR EACH order WHERE ship-date <> ?:
     nrecs = nrecs + 1.
     FOR EACH order-line OF order:
         DISPLAY order-num line-num qty price.
        IF del THEN DELETE order-line.
     END.
     IF del THEN DELETE order.
END.
OUTPUT CLOSE.
```

The following example is a startup procedure. It defines a new global variable with the initial value TRUE and uses that variable to determine whether to run an initialization procedure, `r-init.p`, that displays sign-on messages. Then the global variable first-time is set to FALSE. If you restart this procedure during the same session (pressed **STOP**), `r-init.p` does not run again.

The procedure also defines the variable selection for entering menu choices within this procedure:

r-dfvar3.p

```
DEFINE NEW GLOBAL SHARED VARIABLE first-time
  AS LOGICAL INITIAL TRUE.
DEFINE VARIABLE selection AS INTEGER FORMAT "9"
  LABEL "Selection".

IF first-time THEN DO:
  RUN r-init.p.
  first-time = false.
END.
FORM
    "    MAIN MENU          " SKIP(1)
    "1 - Accounts Payable   " SKIP
    "2 - Accounts Receivable" WITH CENTERED ROW 5 FRAME menu.
REPEAT:
    VIEW FRAME menu.
    UPDATE selection AUTO-RETURN WITH FRAME sel
      CENTERED ROW 12 SIDE-LABELS.
    IF selection = 1 THEN DO:
      HIDE FRAME menu.
      HIDE FRAME sel.
      RUN apmenu.p.
    END.
    ELSE IF selection = 2 THEN DO:
        HIDE FRAME menu.
        HIDE FRAME sel.
        RUN armenu.p.
      END.
      ELSE DO:
        MESSAGE "Invalid selection. Try again".
        UNDO, RETRY.
      END.
END.
```

The following procedure finds the day of the week of a date the user enters. The procedure defines an array with seven elements and uses the INITIAL option to define the initial value of each element in the array.

r-dfvar4.p

```
DEFINE VARIABLE dow AS CHARACTER FORMAT "x(9)" EXTENT 7
  INITIAL ["Sunday", "Monday", "Tuesday", "Wednesday",
  "Thursday", "Friday", "Saturday"].

DEFINE VARIABLE dob AS DATE INITIAL TODAY.

REPEAT WITH SIDE-LABELS 1 DOWN CENTERED ROW 10 TITLE "Date of Birth":
  DISPLAY SKIP(1).
  UPDATE dob LABEL "Enter date of birth".
  DISPLAY dow[WEEKDAY(dob)] LABEL "It was a".
END.
```

The following example defines a variable with a VIEW-AS phrase and a Trigger phrase:

r-defsel.p

```
DEFINE VARIABLE i AS INTEGER NO-UNDO.

DEFINE VARIABLE clubs AS CHARACTER
  VIEW-AS SELECTION-LIST SIZE 20 BY 5 MULTIPLE
    SCROLLBAR-VERTICAL NO-DRAG
    LIST-ITEMS "One Iron", "Two Iron", "Three Iron", "Four Iron",
               "Five Iron", "Six Iron", "Seven Iron", "Eight Iron",
               "Nine Iron", "Pitching Wedge"
  LABEL "Golf Clubs Available"
  TRIGGERS:
    ON GO
      DO:
        IF SELF:SCREEN-VALUE <> "" THEN
        DO i = 1 TO NUM-ENTRIES(SELF:SCREEN-VALUE) :
           DISPLAY ENTRY(i, SELF:SCREEN-VALUE) FORMAT "X(16)"
            WITH FRAME clubs-sel CENTERED
              NUM-ENTRIES(SELF:SCREEN-VALUE) + 1 DOWN
              TITLE "Clubs Selected" USE-TEXT.
          DOWN 1 WITH FRAME clubs-sel.
        END.
      END.
  END TRIGGERS.

ENABLE clubs WITH FRAME get-info TITLE "Select the Desired Club(s)".

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- You can use the DEFINE VARIABLE statement anywhere in a procedure. However, all references to the variable must appear after the DEFINE VARIABLE statement that defines it.
- You cannot define a variable as a BLOB or CLOB field. You can define a variable using their MEMPTR and LONGCHAR counterparts, respectively.
- Defining a LONGCHAR variable supports the same options as a CHARACTER variable, except for the FORMAT option and all VIEW-AS options except VIEW-AS EDITOR LARGE.
- You should use the CASE-SENSITIVE option only when it is important to distinguish between uppercase and lowercase values entered for a character variable. For example, use CASE-SENSITIVE to define a variable for a part number that contains mixed uppercase and lowercase characters.
- After you use the DEFINE NEW GLOBAL SHARED VARIABLE statement to create a global shared variable, use the DEFINE SHARED VARIABLE statements in other procedures to access that variable.
- You cannot define the same global variable twice in the same OpenEdge session. If you try, and the definitions of the two variables do not match, Progress returns an error. If the definitions of the two variables match, Progress disregards the second variable you tried to define (if you are rerunning a startup procedure).
- Changes made to variables when there is no active transaction are not undone when a block is undone.
- When a procedure names and uses a shared variable:
  - Progress searches through the calling chain of procedures looking for the most recent DEFINE NEW SHARED VARIABLE statement that created that shared variable.
  - If no DEFINE NEW SHARED VARIABLE statement is found, Progress searches for a DEFINE NEW GLOBAL SHARED VARIABLE statement that created the shared variable.

  - If the procedure that names the shared variable is called from a trigger or internal procedure that is part of a persistent procedure context, the persistent context is also checked for the most recent DEFINE NEW SHARED VARIABLE or DEFINE NEW GLOBAL SHARED VARIABLE statement at the point in the calling chain where the trigger or internal procedure is executed.

  - If Progress finds one of these statements, it does not search any further for other statements that might have defined the same variable as NEW or NEW GLOBAL.

- Progress checks the definition of a SHARED variable against that of the corresponding NEW SHARED or NEW GLOBAL SHARED variable. The data types and array extents must match. If the FORMAT, LABEL and DECIMALS specifications are not the same, each procedure uses its individual specification. The DEFINE NEW statement determines if a shared variable is NO-UNDO.

- A SHARED variable remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the variable as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED variable, Progress includes the persistent procedure in the resolution of the corresponding NEW SHARED variable as though the procedure were on the procedure call stack.

- If an application with several procedures defines a NEW SHARED variable with the same name in each procedure, Progress creates a different instance of the NEW SHARED variable in each procedure. This behavior supports recursive procedures and bill-of-materials applications.

- You can neither define a SHARED or NEW SHARED variable, nor access such a variable defined in a procedure file, from within a class definition (`.cls`) file. If you do, Progress generates a compilation error. However, multiple procedure (`.p`) files can define and access an object reference variable for a class object instance as a NEW SHARED or NEW GLOBAL SHARED variable. In this case, the object reference variables must be defined for the same class (not a subclass or a super class) in all procedures that use them.

- For SpeedScript, the following options are invalid: BGCOLOR, CONTEXT-HELP-ID, DCOLOR, FONT, FGCOLOR, MOUSE-POINTER, PFCOLOR, and *`view-as-phrase`*.

See also DEFINE BUFFER statement, RUN statement, Trigger phrase, VIEW-AS phrase

DEFINE WORK-TABLE statement

Defines a work table (a temporary table stored in memory) for use within a procedure or within several procedures.

Syntax

```
DEFINE [ [ NEW ] SHARED ] [ PRIVATE ]
  { WORK-TABLE | WORKFILE } work-table-name [ NO-UNDO ]
  [ LIKE tablename [ VALIDATE ] ]
  [ FIELD field-name { AS data-type | LIKE field }
    [ field-options ] ] ...
```

`NEW SHARED { WORK-TABLE | WORKFILE } work-table-name`

Defines and identifies a work table to be shared by a procedure called directly or indirectly by the current procedure. The called procedure must name the same work table in a DEFINE SHARED WORK-TABLE statement. The WORKFILE keyword is allowed for backward compatibility only; using WORK-TABLE or WORKFILE has the same effect.

`SHARED { WORK-TABLE | WORKFILE } work-table-name`

Defines and identifies a work table that was defined by another procedure that used the DEFINE NEW SHARED WORK-TABLE statement. The WORKFILE keyword is allowed for backward compatibility only; using WORK-TABLE or WORKFILE has the same effect.

`[ PRIVATE ] { WORK-TABLE | WORKFILE } work-table-name`

Defines and identifies a work table as a data member for a class, and optionally specifies an access mode for that data member. The WORKFILE keyword is allowed for backward compatibility only; using WORK-TABLE or WORKFILE has the same effect. Do not specify the access mode when defining a work table for a method within a class.

PRIVATE data members can be accessed only by the defining class. The default access mode is PRIVATE.

Note: This option is applicable only when defining a data member for a class in a class definition (`.cls`) file.

{ WORK-TABLE | WORKFILE } *work-table-name*

Defines and identifies a work table whose records are available only within the current procedure, or a method within a class.

The WORKFILE keyword is allowed for backward compatibility only; using WORK-TABLE or WORKFILE has the same effect.

NO-UNDO

Specifies that Progress should not restore the record to its prior condition when a work table record is changed during a transaction and the transaction is undone. If you do not want the work table record undone even if it has changed during a transaction, use the NO-UNDO option with the DEFINE WORK-TABLE statement. NO-UNDO work tables are efficient; use them whenever possible.

LIKE *table-name*

Indicates the name of a table whose characteristics you want to use for the work table you are defining. All of the fields in this base table are also in the work table. If you reference a database table with the LIKE option, the database containing that table must be connected at compile time. It need not be connected at run time.

If more than one connected database contains a table named *table-name*, you must qualify the table name with the database name. See the Record phrase description for more information.

HELP options are inherited from the *table-name*. Validate options are inherited only if the VALIDATE keyword is used.

VALIDATE

The work table fields inherit, from the dictionary, validation expressions and validation messages from the database table, *table-name*.

FIELD *field-name*

Identifies the name of a field in the work table.

AS *datatype*

Indicates the data type of the field or variable you are defining. The data types are CHARACTER, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INTEGER, LOGICAL, RAW, RECID, ROWID, and WIDGET-HANDLE.

LIKE *field*

Indicates the name of the variable, database field, temporary table field, or work table field whose characteristics you want to use for the work table field you are defining. If you name a variable with this option, you must have defined that variable earlier in the procedure. The work table field inherits the data type, extents, format, initial value, label, and column label of the *field*. You can override specific values by using the FORMAT, LABEL, INITIAL, DECIMALS, and EXTENT options. If you do not use these options, the field or variable takes on the characteristics of the variable or database field you name.

If you reference a database field in the LIKE option, the database containing that field must be connected at both compile time and run time. Therefore, use the LIKE option with caution.

field-options

Specifies options for the temporary table field. Any options you specify override any options inherited through the LIKE option. This is the syntax for *field-options*:

```
{ [ BGCOLOR expression ]
  [ COLUMN-LABEL label ]
  [ DCOLOR expression ]
  [ DECIMALS n ]
  [ EXTENT n ]
  [ FONT expression ]
  [ FGCOLOR expression ]
  [ FORMAT string ]
  [ INITIAL
      { constant | { [ constant [ , constant ] ... ] } } ]
  [ LABEL label [ , label ] ... ]
  [ MOUSE-POINTER expression ]
  [ [ NOT ] CASE-SENSITIVE ]
  [ PFCOLOR expression ]
  { [ view-as-phrase ] } }
```

Note: You cannot specify a BLOB field, a CLOB field, or an indeterminate array field in a work-table.

For a description of each option, see the DEFINE VARIABLE statement.

Example

The r-wrkfil.p procedure accumulates all balances by state and stores that information for display later. The procedure uses a work table to accomplish this task.

The r-wrkfil.p procedure defines the work table showsales. The work table contains the three fields named region, state, and tot-sales. These fields have all the same characteristics (except labels) as the customer.sales-region, customer.state, and customer.balance fields, respectively.

The first FOR EACH loop in the r-wrkfil.p procedure sorts customers by state. Then it accumulates the balances for each customer by state. When the procedure finds the last customer in a state, it creates a showsales record for that state. The procedure assigns information to the fields in the showsales record. After looking at each customer, the procedure continues to the next FOR EACH statement.

The second FOR EACH statement in the r-wrkfil.p procedure uses the information stored in the showsales table. Because you treat a work table within a procedure the same way you treat a database table, you can perform the same work with the showsales table that you can with a database table.

r-wrkfil.p

```
DEFINE WORK-TABLE showsales
   FIELD region    LIKE salesrep.region LABEL        "Region"
   FIELD state     LIKE customer.state  LABEL        "St"
   FIELD tot-sales LIKE cust.balance    COLUMN-LABEL "Total!Sales".

FOR EACH customer, salesrep OF customer BREAK BY customer.state:
   ACCUMULATE balance (TOTAL by customer.state).
   IF LAST-OF(customer.state) THEN DO:
     CREATE showsales.
     showsales.state = customer.state.
     showsales.tot-sales = ACCUM TOTAL BY customer.state balance.
     showsales.region = salesrep.region.
   END.
END.

FOR EACH showsales BREAK BY showsales.region BY showsales.state:
   IF FIRST-OF (showsales.region)
      THEN DISPLAY showsales.region.
   DISPLAY showsales.state tot-sales (TOTAL BY showsales.region).
END.
```

Notes

- You cannot perform a unique find on a work table. When finding records in a work table, you must use FIRST, LAST, NEXT, or PREV with the FIND statement, unless you are finding a record using its ROWID.
- You cannot define a field in a work table with the MEMPTR data type, but you can define a work table field as ROWID or RAW.
- You cannot define shared objects, work tables, or temporary tables within an internal procedure, a method in a class, or a user-defined function.
- Progress disregards the following when used in conjunction with a work table:
  - The VALIDATE option on a DELETE statement.
  - The SHARE-LOCK, EXCLUSIVE-LOCK, and NO-LOCK options used with the FIND or FOR statements.
  - The NO-WAIT option on the FIND statement.
- When you use the AMBIGUOUS function in conjunction with a work table, the function always returns a value of FALSE.
- Complete work table definitions must be included in a DEFINE SHARED WORK-TABLE statement and shared work tables must be defined identically.
- These are the differences between work tables and regular database tables:
  - Progress does not use the OpenEdge database manager (and server for multi-user systems) when working with work tables.
  - If you do not explicitly delete the records in a work table, Progress discards those records, and the work table, at the end of the procedure that initially defined the work table.
  - Users do not have access to each other’s work tables.

- Because you cannot index a work table, Progress uses the following rules for storing records in a work table:
  - If you create a series of work table records without doing any other record operations, Progress orders the newly created records in the order they were entered.
  - If you use the FIND PREV statement at the beginning of a work table and then create a work table record, Progress stores that record at the beginning of the work table.
  - When you use the FIND statement to find a work table record and then use the CREATE statement to create a new work table record, Progress stores that new record after the record you just found.
- Data handling statements that cause Progress to automatically start a transaction for a regular table will not cause Progress to automatically start a transaction for a work table. To start a transaction for operations involving a work table, Use the TRANSACTION keyword.
- Work tables are private:
  - Even if two users define work tables with the same name, the work tables are private; one user cannot see records the other user has created.
  - If two procedures run by the same user define work tables with the same name, Progress treats those work tables as two separate tables unless the SHARED option is included in both procedures.
- DEFINE SHARED WORK-TABLE does not automatically provide a shared buffer. If you want to use a shared buffer with a shared work table, you must define that buffer.
- Work table records are built in 64-byte sections. Approximately the first 60 bytes of each record are taken up by record specification information (or a record header). That is, if a record is 14 bytes long, it will be stored in two 64-byte sections, using the first 60 bytes as a record header. If the record is 80 bytes long, it will fit into three 64-byte sections. The first part contains 60 bytes of header information plus the first 4 bytes of the record. The second section contains 64 bytes of the record. And the last section contains the remaining record bytes.

- The NO-UNDO option in a work table definition overrides a transaction UNDO for CREATE, UPDATE, DELETE, and RELEASE statements accessing the work table, regardless of whether these statements are executed before or during the transaction block that is undone.
- A transaction UNDO overrides a FIND statement accessing a work table defined with the NO-UNDO option, regardless of whether the find is executed before or during the transaction that is undone.

  You should use the CASE-SENSITIVE option only when it is important to distinguish between uppercase and lowercase values entered for a character field. For example, use CASE SENSITIVE to define a field for a part number that contains mixed upper case and lowercase characters.
- A SHARED work table remains in scope for an instance of a persistent procedure until the instance is deleted. This is true even if the original procedure that defined the work table as NEW SHARED goes out of scope while the procedure instance remains persistent.

  If a trigger or internal procedure of a persistent procedure executes an external subprocedure that defines a SHARED work table, Progress includes the persistent procedure in the resolution of the corresponding NEW SHARED work table as though the procedure were on the procedure call stack.
- You cannot define a SHARED or NEW SHARED work table in a class definition (`.cls`) file. If you do, Progress generates a compilation error.
- You can specify a join between a temporary table or work table and any appropriate table using the OF keyword. The two tables must contain a commonly named field that participates in a unique index for at least one of the tables. For more information on table joins see the Record phrase reference entry.
- See *OpenEdge Development: Progress 4GL Handbook* for information on work tables and temporary tables.

See also { } Argument reference, { } Include file reference, CREATE statement, DEFINE BUFFER statement, DEFINE TEMP-TABLE statement, FIND statement, Format phrase, RUN statement

DEFINE WORKFILE statement

See the DEFINE WORK-TABLE statement reference entry.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEFINE [ [ NEW ] SHARED ]
  { WORK-TABLE | WORKFILE } work-table-name
  [ NO-UNDO ]
  [ LIKE tablename ]
  [ FIELD field-name
      {     AS data-type
          | LIKE field
      }
      [ field-options ]
  ] ...
```

DEFINED preprocessor function

Returns the status of a preprocessor name or include file argument name. You can use the DEFINED function only within a preprocessor &IF expression.

Syntax

```
DEFINED ( name )
```

name

Preprocessor name or include file argument name whose status you want to check. You do not specify *name* as a preprocessor name reference or include file argument reference. That is, it is not quoted and does not appear in the reference form, {&*name*}. For example if you had a preprocessor name MAX-EXPENSE, the argument would appear as follows:

```
DEFINED(MAX-EXPENSE)
```

Note

This function returns a value of 1 if the argument was a name defined with the &GLOBAL-DEFINE directive; a value of 2 if the argument was passed as an include file parameter; and a value of 3 if the argument was a name defined with the &SCOPED-DEFINE directive. If the argument was not defined and was not an include file parameter, then this function returns a value of 0. The value returned refers to the definition that is current at the point of the call.

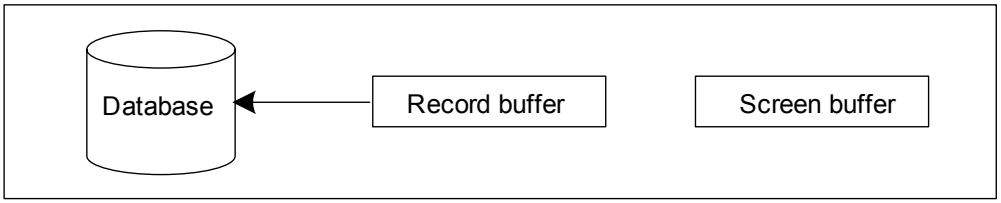
See also

{ } Include file reference, &GLOBAL-DEFINE preprocessor directive, &SCOPED-DEFINE preprocessor directive

DELETE statement

Removes a record from a record buffer and from the database.

Data movement



Syntax

```
DELETE record
  [ VALIDATE ( condition , msg-expression ) ]
  [ NO-ERROR ]
```

record

The name of a record buffer. You can delete a record only after it has been put into a record buffer by a CREATE, FIND, FOR EACH, or INSERT statement.

If you define an alternate buffer for a table, you can delete a record from that buffer by using the name of the buffer with the DELETE statement.

To delete a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

VALIDATE (*condition*, *msg-expression*)

Use the VALIDATE option to specify a logical value that allows the deletion of a record when TRUE, but does not allow the deletion of a record when FALSE.

The *condition* is a Boolean expression (a constant, field name, variable name, or expression) with a value of TRUE or FALSE.

The *msg-expression* is the message you want to display if the *condition* is FALSE. You must enclose *msg-expression* in quotation marks ("").

You can also describe delete validation criteria for a table in the Data Dictionary. To suppress the Data Dictionary delete validation criteria for a table, use this VALIDATE option.

```
VALIDATE(TRUE,"")
```

If you use the DELETE statement to delete a record in a work table, Progress disregards any VALIDATE option you use with the DELETE statement.

NO-ERROR

Specifies that any errors that occur when you try to delete the record are suppressed. After the DELETE statement completes, you can check the ERROR-STATUS system handle for information on any errors that might have occurred.

Examples

The r-delet.p procedure deletes all the records in the customer table.

r-delet.p

```
FOR EACH customer:
  DELETE customer.
END.
```

The r-delet2.p procedure prompts the user for a customer number and then displays the name of that customer. It then prompts the user to press y to confirm the deletion of the customer record. The user's response is stored in the del variable. If the value of the del variable is y, the procedure deletes the customer record.

r-delet2.p

```
DEFINE VARIABLE del AS LOGICAL FORMAT "y/n".
REPEAT:
  PROMPT-FOR customer.cust-num.
  FIND customer USING cust-num.
  DISPLAY name.
  del = no.
  UPDATE del LABEL "Enter ""y"" to confirm delete".
  IF del THEN DELETE customer.
END.
```

The `r-delval.p` procedure prompts the user for a customer number. The procedure displays the name of the customer and prompts the user: Do you want to delete this customer? If the user answers no, the procedure prompts the user for another customer number. If the user answers yes, the procedure checks whether the customer has orders, using the VALIDATE option. If they do have orders, the procedure displays this message: This customer has outstanding orders and cannot be deleted. If the customer has no orders, the procedure deletes the customer.

r-delval.p

```
DEFINE VARIABLE ans AS LOGICAL.

REPEAT WITH 1 DOWN:
  PROMPT-FOR customer.cust-num.
  FIND customer USING customer.cust-num.
  DISPLAY name.
  ans = no.
  DISPLAY "Do you want to delete this customer ?"
    WITH FRAME f-query.
  UPDATE ans WITH FRAME f-query NO-LABELS.
  IF ans
  THEN DELETE customer
    VALIDATE(NOT(CAN-FIND(order OF customer)),
    "This customer has outstanding orders and cannot be deleted.").
END.
```

Notes

- When you run procedures that delete large numbers of records (for example, a month-end table purge), the process runs much faster if you use the No Crash Protection (-i) parameter in single-user mode. (You must back up your database before using this option.) See *OpenEdge Deployment: Startup Command and Parameter Reference* for more information on startup parameters.
- Deleting records does not change the amount of space the database takes up on the disk. Progress re-uses ROWIDs. It does not delete the ROWID when a record is deleted. To recover disk space, you must dump and reload your database.
- The DELETE statement causes any related database DELETE triggers to execute. All DELETE triggers execute before Progress actually deletes the record. While a DELETE trigger is executing, all FIND requests for the record (even within the trigger) fail, as if the record were already deleted. If a DELETE trigger fails (or executes a RETURN ERROR statement), the corresponding record is not deleted. See *OpenEdge Development: Progress 4GL Handbook* for more information on database triggers.
- If a table has both a DELETE trigger and delete VALIDATION, the DELETE trigger executes before the validation is performed.
- If you have previously retrieved *record* with a field list, the DELETE statement rereads the complete record before deleting it.

See also CREATE statement, FIND statement, FOR statement, INSERT statement

DELETE ALIAS statement

Deletes an alias from the alias table.

Syntax

```
DELETE ALIAS { alias | VALUE ( expression ) }
```

alias

An existing alias. It can be an unquoted string or a quoted string.

VALUE (*expression*)

A character-string expression that evaluates to an existing alias.

Example

This procedure deletes the alias myalias from the alias table:

r-dalias.p

```
DELETE ALIAS myalias.
```

Notes

- If a precompiled program requires an alias and you delete that alias, the program will not run.
- If you try to delete a nonexistent alias, nothing happens.

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DELETE OBJECT statement

Deletes an instance of an object such as a widget, a procedure, a server, a socket, or a class. Deleting the object instance causes all allocated resources associated with the object instance to be returned to the system (except when otherwise noted).

Syntax

```
DELETE OBJECT { handle | object-reference } [ NO-ERROR ]
```

handle

A handle to the object to delete. The handle argument must be a variable of type HANDLE and must contain a valid handle.

If the handle parameter refers to a widget handle, the DELETE OBJECT statement is a synonym for the DELETE WIDGET statement.

If the handle parameter refers to a persistent procedure handle or proxy persistent procedure handle, the DELETE OBJECT statement is a synonym for the DELETE PROCEDURE statement. This statement deletes a local persistent procedure handle immediately. For a proxy persistent procedure handle, this statement deletes the handle immediately unless there is an outstanding asynchronous request on this handle (*handle*:ASYNC-REQUEST-COUNT is greater than zero (0)). If *handle*:ASYNC-REQUEST-COUNT is greater than zero (0), this statement raises the ERROR condition. Otherwise, the statement also sends a request to the AppServer to delete the corresponding remote persistent procedure on the AppServer. If the AppServer is executing any asynchronous requests ahead of it, Progress queues the delete request (as with any asynchronous remote request) until the AppServer is available to handle it.

Note: This same behavior occurs if the remote procedure deletes itself (using DELETE...THIS-PROCEDURE) on the AppServer.

For more information on remote persistent procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

If the handle parameter refers to a server handle, the DELETE OBJECT statement:

- Checks that the handle parameter refers to a valid server handle, and that the handle parameter's CONNECTED attribute is FALSE (no AppServer is connected to it). If one of these checks fails, the statement raises the ERROR condition.
- Deletes the handle immediately, if the server handle is valid, unless there is an outstanding asynchronous request on this handle (*handle*:ASYNC-REQUEST-COUNT is greater than zero (0)). If there is an outstanding asynchronous request, this statement raises the ERROR condition.

Deleting a server handle removes the handle from the server handle chain of the SESSION system handle, and resets SESSION:FIRST-SERVER and SESSION:LAST-SERVER if necessary. This also deletes all of the asynchronous request handles associated with the server and then deletes the server object.

If *handle* refers to an asynchronous request handle, the DELETE OBJECT statement takes one of the following actions:

- If the *handle*:COMPLETE attribute is FALSE, raises the ERROR condition.
- If the *handle*:COMPLETE attribute is TRUE, removes *handle* from the chain of asynchronous request handles referenced by the FIRST-ASYNC-REQUEST and the LAST-ASYNC-REQUEST attributes of the server handle, and deletes *handle*.

If this is a socket handle, the application must disconnect the socket from a port using the DISCONNECT( ) method before a socket object can be deleted. The DELETE OBJECT statement will raise ERROR if an application deletes a socket object that is still associated with a port.

If this is a server socket handle, the application must call DISABLE-CONNECTIONS( ) before a server socket object can be deleted. The DELETE OBJECT statement will raise ERROR if an application deletes a server socket object that is still listening for connections.

object-reference

An object reference for the class object instance to delete. The object reference argument must be an object reference variable, such as one defined using the DEFINE VARIABLE statement with the CLASS option, and must contain a valid object reference.

Note: You can validate an object reference by using the VALID-OBJECT function.

When you delete a class object instance, Progress invokes the destructor method for the class and the destructor method for each class in its inherited class hierarchy, if any. The destructor method is responsible for freeing resources allocated by the object instance when the object is deleted. At this time, the object instance context goes out of scope and all allocated resources are returned to the system. In addition, the object instance is removed from the list of valid object instances referenced by the FIRST-OBJECT and LAST-OBJECT attributes of the SESSION system handle.

If you do not delete a class object instance, the instance remains in memory until the session ends, and Progress does not invoke any destructor methods for the class.

NO-ERROR

Suppresses reporting of errors that occur while DELETE OBJECT executes. Afterwards, you can get information on possible errors by checking the ERROR-STATUS system handle.

Notes

- For more information on working with asynchronous remote procedures and event procedures, see *OpenEdge Application Server: Developing AppServer Applications*.
- For more information on working with socket and server socket objects, see *OpenEdge Development: Programming Interfaces*.

See also DELETE PROCEDURE statement, DELETE WIDGET statement, DESTRUCTOR statement, ERROR-STATUS system handle

DELETE PROCEDURE statement

Deletes an instance of a persistent procedure. The persistent procedure can be local or remote.

Syntax

```
DELETE PROCEDURE proc-handle [ NO-ERROR ]
```

proc-handle

The handle of a local or remote persistent procedure. This is a variable, field, or expression of type HANDLE that contains a valid persistent procedure handle.

For a proxy persistent procedure handle, this statement deletes the handle immediately unless there is an outstanding asynchronous request on this handle (*handle*:ASYNC-REQUEST-COUNT is greater than zero (0)). If *handle*:ASYNC-REQUEST-COUNT is greater than zero (0), this statement raises the ERROR condition. Otherwise, the statement also sends a request to the AppServer to delete the corresponding remote persistent procedure on the AppServer. If the AppServer is executing any asynchronous requests ahead of it, Progress queues the delete request (as with any asynchronous remote request) until the AppServer is available to handle it.

Note: This same behavior occurs if the remote procedure deletes itself (using DELETE...THIS-PROCEDURE) on the AppServer.

For more information on remote persistent procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

NO-ERROR

Specifies that any errors that occur when you try to delete the procedure are suppressed. After the DELETE PROCEDURE statement completes, you can check the ERROR-STATUS system handle for information on any errors that might have occurred.

Example

When you run the following procedure non-persistently, the procedure creates a persistent instance of itself in addition to the non-persistent instance, creating two query windows for the customer table. Choosing the Cancel button in either window causes the instance that owns that window to terminate. If the instance you terminate is persistent, the Cancel button runs an internal procedure that executes the DELETE PROCEDURE statement for that instance as specified by the THIS-PROCEDURE system handle.

r-delprc.p

```
DEFINE QUERY custq FOR customer.
DEFINE BROWSE custb QUERY custq
    DISPLAY name balance phone WITH 10 DOWN.
DEFINE BUTTON bName LABEL "Query on Name".
DEFINE BUTTON bBalance LABEL "Query on Balance".
DEFINE BUTTON bCancel LABEL "Cancel".
DEFINE FRAME CustFrame custb SKIP bName bBalance bCancel.
DEFINE VARIABLE custwin AS WIDGET-HANDLE.

ON CHOOSE OF bName IN FRAME CustFrame DO:
    custwin:TITLE = "Customers by Name".
    OPEN QUERY custq FOR EACH customer BY name.
END.
ON CHOOSE OF bBalance IN FRAME CustFrame DO:
    custwin:TITLE = "Customers by Balance".
    OPEN QUERY custq FOR EACH customer BY balance DESCENDING.
END.

IF THIS-PROCEDURE:PERSISTENT THEN DO:
    THIS-PROCEDURE:PRIVATE-DATA = "Customer Browse".
    CREATE WIDGET-POOL.
END.
CREATE WINDOW custwin
    ASSIGN
        TITLE = IF THIS-PROCEDURE:PERSISTENT
                THEN "Persistent Customer Browser"
                ELSE "Customer Browser"
        SCROLL-BARS = FALSE
        MAX-HEIGHT-CHARS = FRAME CustFrame:HEIGHT-CHARS
        MAX-WIDTH-CHARS = FRAME CustFrame:WIDTH-CHARS.
THIS-PROCEDURE:CURRENT-WINDOW = custwin.
ENABLE ALL WITH FRAME CustFrame.
IF THIS-PROCEDURE:PERSISTENT THEN DO:
    ON CHOOSE OF bCancel IN FRAME CustFrame DO:
        RUN destroy-query.
    END.
END.
ELSE DO:
    RUN r-delprc.p PERSISTENT.
    WAIT-FOR CHOOSE OF bCancel IN FRAME CustFrame.
    DELETE WIDGET custwin.
END.

PROCEDURE destroy-query:
    DELETE PROCEDURE THIS-PROCEDURE.
    DELETE WIDGET-POOL.
END.
```

Notes

- To be valid for deletion, *proc-handle* must reference an active persistent procedure. You can use the VALID-HANDLE function and PERSISTENT procedure attribute to check the validity of *proc-handle*. Thus, both VALID-HANDLE(*proc-handle*) and *proc-handle*:PERSISTENT must be TRUE to delete the specified procedure. If either of these expressions is FALSE, the DELETE PROCEDURE statement raises the ERROR condition.
- When you delete a persistent procedure instance, its context goes out of scope and all allocated resources are returned to the system. In addition, it is removed from the chain of persistent procedures referenced by the FIRST-PROCEDURE and LAST-PROCEDURE attributes of the SESSION system handle.
- If you delete a persistent procedure instance while executing statements within that procedure, the DELETE PROCEDURE statement pends until the largest executing block in the persistent procedure terminates. Thus, if the DELETE PROCEDURE occurs while the main procedure block is executing (when the persistent procedure is called), the procedure is deleted when the procedure returns (as if it were non-persistent). If the DELETE PROCEDURE occurs during execution of a trigger or execution of an internal procedure that is called from another external procedure, the procedure is deleted after the trigger block or internal procedure returns. Note that while the delete is pending, the persistent procedure remains valid in the persistent procedure chain.
- The DELETE PROCEDURE statement disconnects any local buffers established by the procedure. In addition, any buffers passed as parameters to a persistent procedure are treated as local buffers. While all cursor positioning established on these buffers by the persistent procedure is lost, there is no affect on the original buffers passed as parameters from the caller. Note that all buffers are validated before being disconnected (which might cause database write triggers to execute). If the validation fails, the DELETE PROCEDURE statement raises the ERROR condition and pends the deletion until the validation succeeds and all database write triggers have completed.
- For more information on working with asynchronous remote procedures and event procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

See also RUN statement, THIS-PROCEDURE system handle, VALID-HANDLE function

DELETE WIDGET statement

Deletes one or more dynamic widgets.

Syntax

```
DELETE WIDGET widget-handle [ widget-handle ] ...
```

widget-handle

The handle of a dynamic widget.

Example In the following example, the DELETE WIDGET statements deletes the dynamic button that you select:

r-delwid.p

```
DEFINE VARIABLE wh-tmp AS WIDGET-HANDLE.
DEFINE VARIABLE i      AS INTEGER NO-UNDO.

DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.
DEFINE BUTTON b_make LABEL "Make Buttons".

DEFINE FRAME butt-frame
  b_make b_quit
  WITH CENTERED ROW 2.

DEFINE FRAME new-buttons WITH WIDTH 48 CENTERED TITLE "New Buttons".

FRAME new-buttons:HEIGHT-CHARS  = 3.

ON CHOOSE OF b_make IN FRAME butt-frame
DO:
  DISABLE b_make WITH FRAME butt-frame.
  DO i = 1 TO 10:
    CREATE BUTTON wh-tmp
      ASSIGN FRAME  = FRAME new-buttons:HANDLE
             COLUMN = i * 4
             LABEL  = STRING(i)
             SENSITIVE = TRUE
             VISIBLE = TRUE
      TRIGGERS:
        ON CHOOSE
           PERSISTENT RUN del-self.
      END.
  END.
END.

ENABLE b_make b_quit WITH FRAME butt-frame.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.

PROCEDURE del-self:
  IF SELF:PREV-SIBLING = ?  AND SELF:NEXT-SIBLING = ? THEN
  DO:
    HIDE FRAME new-buttons.
    ENABLE b_make WITH FRAME butt-frame.
  END.
  MESSAGE "Deleting Widget, Button" SELF:LABEL.
  DELETE WIDGET SELF.
END.
```

Notes

- If you do not explicitly delete a dynamically created widget, it is deleted when its widget pool is deleted. If you do not create a new unnamed widget pool and do not explicitly specify a named widget pool when you create the widgets, all dynamic widgets are placed in the session pool. The session pool is not deleted until the OpenEdge session that created it ends.
- If *widget-handle* refers to a control-frame, any ActiveX control associated with the widget is also deleted. For more information on ActiveX support in Progress, see *OpenEdge Development: Programming Interfaces*.
- For SpeedScript, use with buffer-field, buffer-object, buffer, and query-object handles.

See also CREATE widget statement

DELETE WIDGET-POOL statement

Deletes a defined widget pool.

Note: Does not apply to SpeedScript programming.

Syntax

```
DELETE WIDGET-POOL [ pool-name ] [ NO-ERROR ]
```

pool-name

The name of a defined dynamic widget pool. If you omit *pool-name*, the statement deletes the unnamed pool most recently created in the current or a calling procedure.

NO-ERROR

Suppresses error messages if the specified widget pool does not exist. You can then test for the ERROR condition to verify that the widget pool does not exist.

Example

The following example creates a named widget pool and lets you add buttons to it. When you choose Delete Buttons, the widget pool is deleted. (Therefore all the buttons in the pool are also deleted.) Similarly, when you choose Quit to exit the procedure the widget pool is also deleted. Because the pool is persistent, it remains allocated for the rest of your session if you do not delete it.

r-widpl.p

```
DEFINE VARIABLE wh AS WIDGET-HANDLE.
DEFINE BUTTON b_create LABEL "Create Button".
DEFINE BUTTON b_del    LABEL "Delete Buttons".
DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE
    DO:
      IF VALID-HANDLE(wh) THEN DELETE WIDGET-POOL "new-buttons".
      QUIT.
    END.
  END.

DEFINE FRAME butt-frame
  b_create b_del b_quit
  WITH ROW SCREEN-LINES - 2.
DEFINE FRAME new-buttons
  WITH SIZE 76 BY 11 CENTERED ROW 2 TITLE "New Buttons".

ON CHOOSE OF b_create IN FRAME butt-frame
DO:
  STATUS INPUT "Press RETURN to select a new button".
  IF wh = ? OR NOT VALID-HANDLE(wh) THEN
    CREATE WIDGET-POOL "new-buttons" PERSISTENT.
  CREATE BUTTON wh IN WIDGET-POOL "new-buttons"
    ASSIGN FRAME = FRAME new-buttons:HANDLE
      ROW = RANDOM(2, 9)
      COLUMN = RANDOM(2, 58)
      LABEL = "BUTTON " + STRING(etime)
      SENSITIVE = TRUE
      VISIBLE = TRUE
      MOVABLE = TRUE
      TRIGGERS:
        ON CHOOSE PERSISTENT RUN dispmsg.
      END.
END.
ON CHOOSE OF b_del IN FRAME butt-frame
DO:
  IF VALID-HANDLE(wh) THEN DELETE WIDGET-POOL "new-buttons".
  STATUS INPUT.
END.ENABLE b_create b_del b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.

PROCEDURE dispmsg:
  MESSAGE "You chose button " SELF:LABEL.
END.
```

Notes

- When you delete a widget pool, all widgets in that pool are automatically deleted.
- If you do not delete a non-persistent widget pool, it is deleted when the procedure that created it ends. If you do not delete a persistent widget pool, it is deleted when the session ends.
- All named widget pools are globally scoped. While a named widget pool is allocated, any procedure within the same process can access that widget pool. If you try to delete a named widget pool that does not exist, Progress raises the ERROR condition.

See also CREATE WIDGET-POOL statement, DELETE WIDGET statement

DESTRUCTOR statement

Defines a destructor method for a class. Progress invokes this destructor method when the object is deleted using the DELETE OBJECT statement.

Note: This statement is applicable only when used in a class definition (`.cls`) file.

Syntax

```
DESTRUCTOR PUBLIC class-name ( ):

destructor-body
```

PUBLIC

Specifies the access mode for this destructor method. The access mode for a destructor method is always PUBLIC.

A PUBLIC destructor method can be accessed indirectly by the defining class, any of its inheriting classes, and any class or procedure that instantiates the class object (that is, through an object reference) by deleting the object instance using the DELETE OBJECT statement.

class-name

The name of the class this method destroys. This name must match the class name portion of the type name for the class (that is, the name of the class definition file excluding the `.cls` extension and any package path information).

destructor-body

The body of the destructor definition. Define the destructor body using the following syntax:

```
      .
      .
      .
method-logic
      .
      .
      .
END [ DESTRUCTOR ].
```

method-logic

The logic of the destructor method, which can contain any Progress 4GL statements currently allowed within a PROCEDURE block including class-related statements, but excluding the RETURN ERROR statement. The method's logic must not reference, either directly or indirectly, statements that block I/O (namely, the CHOOSE, INSERT, PROMPT-FOR, READKEY, SET, UPDATE, and WAIT-FOR statements).

This method typically contains logic to release system resources used by the class object instance.

END [DESTRUCTOR]

Specifies the end of the destructor body definition. You must end the destructor body definition with the END statement.

Example

The following example shows the definition of a destructor method:

```
DESTRUCTOR PUBLIC CustObj( ):

  EMPTY TEMP-TABLE ttCust.

END DESTRUCTOR.
```

Notes

- You can terminate a DESTRUCTOR statement with either a period (.) or a colon (:).
- A complete destructor method definition must begin with the DESTRUCTOR statement and end with the END statement.
- A destructor method has no parameters and no return value.
- You never explicitly invoke the destructor method to delete a class object instance. The method is implicitly invoked when the object instance is destroyed by the DELETE OBJECT statement.

See also

CLASS statement, CONSTRUCTOR statement, DELETE OBJECT statement

DICTIONARY statement

Runs the OpenEdge Data Dictionary.

Note: Does not apply to SpeedScript programming.

Syntax

```
DICTIONARY
```

Example

This procedure runs the Data Dictionary if the user answers yes to a prompt:

r-dict.p

```
DEFINE VARIABLE ans AS LOGICAL.

DISPLAY "Do you want to access the Dictionary?"
  WITH ROW 7 COLUMN 20 NO-LABELS.
UPDATE ans.
IF ans THEN DICTIONARY.
```

Notes

- The DICTIONARY statement is equivalent to RUN dict.p: it runs the Progress procedure called `dict.p`. Progress uses the regular search rules to find the dictionary procedure. The dictionary procedure is part of the Progress system software.
- Progress Query/Run-Time provides a restricted version of the Data Dictionary.
- For more information on the Data Dictionary, see its on-line help.

DISABLE statement

Disables input for one or more field-level and child frame widgets within a frame that were previously enabled with the ENABLE statement. Disabling a widget prevents the user from providing input to the widget, but does not remove it from the display.

Syntax

```
DISABLE [ UNLESS-HIDDEN ]
  {     ALL [ EXCEPT field ... ]
    |  { field [ WHEN expression ] } ...
  }
  { [ frame-phrase ] }
```

UNLESS-HIDDEN

Restricts DISABLE to fields whose HIDDEN attribute is FALSE.

ALL [EXCEPT *field* ...]

Specifies that all field-level widgets for a frame should be disabled, except those that you optionally specify.

field [WHEN *expression*]

A field-level widget to be disabled. If you use the WHEN option, then the field is disabled only if *expression* is TRUE when the DISABLE statement is executed. The *expression* must evaluate to a LOGICAL value.

frame-phrase

The frame that contains the widgets to disable. If you omit *frame-phrase*, the default frame for the current block is assumed.

You cannot use the IN WINDOW option of the frame phrase within a DISABLE statement. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

In the following example, the cust-num field and the Quit button are initially active. When you press **GO** in the cust-num field, that field becomes disabled and the Save and Undo buttons and the credit-limit field are enabled. If you choose either the Save or Undo button, those buttons and the credit-limit field are again disabled and the cust-num field is enabled again.

r-enable.p

```
DEFINE VARIABLE ok AS LOGICAL NO-UNDO.

DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.
DEFINE BUTTON b_save LABEL "Save".
DEFINE BUTTON b_undo LABEL "Undo".DEFINE FRAME butt-frame
  b_save b_undo b_quit
  WITH CENTERED ROW SCREEN-LINES - 1.

FORM
  customer
WITH FRAME cust-info SIDE-LABELS CENTERED
     TITLE "Update Customer Credit Limit".

ON CHOOSE OF b_save, b_undo IN FRAME butt-frame
DO:
  DISABLE b_save b_undo WITH FRAME butt-frame.
  DISABLE customer.credit-limit WITH FRAME cust-info.
  ENABLE customer.cust-num WITH FRAME cust-info.
  IF SELF:LABEL = "save" THEN
    ASSIGN FRAME cust-info customer.credit-limit.
  CLEAR FRAME cust-info NO-PAUSE.
  APPLY "ENTRY" TO customer.cust-num IN FRAME cust-info.
END.ON GO OF customer.cust-num IN FRAME cust-info
DO:
  FIND customer USING customer.cust-num EXCLUSIVE NO-ERROR.
  IF AVAILABLE(customer) THEN
  DO:
    DISABLE customer.cust-num WITH FRAME cust-info.
    ENABLE customer.credit-limit WITH FRAME cust-info.
    ENABLE ALL WITH FRAME butt-frame.
    DISPLAY customer WITH FRAME cust-info.
  END.
  ELSE
  DO:
     MESSAGE "No Customer Record exist for customer number"
             INPUT customer.cust-num ", Please re-enter."
       VIEW-AS ALERT-BOX WARNING BUTTONS OK-CANCEL UPDATE OK.
    IF NOT ok THEN
      APPLY "CHOOSE" TO b_quit IN FRAME butt-frame.
  END.
END.

ENABLE customer.cust-num WITH FRAME cust-info.
ENABLE b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame
   FOCUS customer.cust-num IN FRAME cust-info.
```

Note If you invoke the DISABLE statement for the parent frame of a frame family, the field representation widgets and descendant frames owned by the parent frame are all disabled. However, the field representation widgets of the descendant frames remain enabled and appear sensitive, although they cannot accept input. To disable field representation widgets in the descendant frames and make them appear insensitive, you must invoke DISABLE statements for each of the descendant frames.

See also ENABLE statement, WAIT-FOR statement

DISABLE TRIGGERS statement

Disables database triggers before you perform a dump or load procedure. You must have CAN-DUMP and CAN-LOAD permissions on the table for which you want to disable the triggers.

Syntax

```
DISABLE TRIGGERS FOR { DUMP | LOAD } OF table-name
  [ ALLOW-REPLICATION ]
```

DUMP

Disabling triggers for DUMP disables the trigger associated with the FIND event for the named table.

LOAD

Disabling triggers for LOAD disables all triggers associated with the CREATE, WRITE, and ASSIGN events for the named table.

table-name

The name of the table for which you want to disable the triggers. You can name only one table.

ALLOW-REPLICATION

Tells DISABLE TRIGGERS to disable only CREATE, DELETE, ASSIGN, and WRITE triggers, and not REPLICATION-CREATE, REPLICATION-DELETE, and REPLICATION-WRITE triggers.

For more information on database replication, see the reference entry for the RAW-TRANSFER statement in this book, and *OpenEdge Data Management: Database Administration*.

Example

The following example lets you dump or load the contents of a database table. The procedure uses the DISABLE TRIGGERS statement to disable the appropriate triggers before each dump or load operation.

r-dstrig.p *(1 of 2)*

```
DEFINE SUB-MENU file
  MENU-ITEM viewit LABEL "&View Data"
  MENU-ITEM dumpit LABEL "&Dump Data"
  MENU-ITEM loadit LABEL "&Load Data".
  MENU-ITEM exit   LABEL "E&xit".

DEFINE MENU mbar MENUBAR
  SUB-MENU file LABEL "&File".

DEFINE BUTTON b_more LABEL "Next".
DEFINE BUTTON b_exit LABEL "Cancel".

DEFINE FRAME cust-frame
  customer.cust-num SKIP
  customer.name  SKIP
  customer.phone SKIP
  b_more b_exit
  WITH CENTERED SIDE-LABELS ROW 3.

DEFINE STREAM cust.

DEFINE VARIABLE i AS INTEGER NO-UNDO.

PAUSE 0 BEFORE-HIDE.

ON CHOOSE OF b_exit IN FRAME cust-frame
DO:
  HIDE FRAME cust-frame NO-PAUSE.
  DISABLE ALL WITH FRAME cust-frame.
  LEAVE.
END.

ON CHOOSE OF b_more IN FRAME cust-frame
DO:
  FIND NEXT customer NO-LOCK NO-ERROR.
  IF NOT AVAILABLE(customer) THEN
    RETURN.
  DISPLAY customer.cust-num customer.name customer.phone
    WITH FRAME cust-frame.
END.
```

r-dstrig.p *(2 of 2)*

```
ON CHOOSE OF MENU-ITEM viewit
DO:
  ENABLE ALL WITH FRAME cust-frame.
  FIND FIRST customer NO-LOCK NO-ERROR.
  DISP customer.cust-num customer.name customer.phone
    WITH FRAME cust-frame.
END.

ON CHOOSE OF MENU-ITEM dumpit
DO:
  DISABLE TRIGGERS FOR DUMP OF customer.
  i = 1.
  SESSION:IMMEDIATE-DISPLAY = TRUE.
  OUTPUT STREAM cust TO "customer.d".
  FOR EACH customer NO-LOCK:
    EXPORT STREAM cust customer.
    DISP i LABEL "Records Processed"
      WITH FRAME rec-info SIDE-LABELS ROW SCREEN-LINES / 2 CENTERED.
    i = i + 1.
    PROCESS EVENTS.
  END.
  SESSION:IMMEDIATE-DISPLAY = FALSE.
  OUTPUT STREAM cust CLOSE. /*
  APPLY "ENTRY" TO b_quit IN FRAME butt-frame. */
END.

ON CHOOSE OF MENU-ITEM loadit
DO:
  DISABLE TRIGGERS FOR LOAD OF customer.
  INPUT FROM "customer.d".
  SESSION:IMMEDIATE-DISPLAY = TRUE.
  REPEAT:
    CREATE customer.
    IMPORT customer.
    DISP i LABEL "Records Processed"
      WITH FRAME rec-info SIDE-LABELS ROW SCREEN-LINES / 2 CENTERED.
    i = i + 1.
    PROCESS EVENTS.
  END.
  INPUT CLOSE.
  SESSION:IMMEDIATE-DISPLAY = FALSE.
END.

IF NOT RETRY THEN
   ASSIGN CURRENT-WINDOW:MENUBAR = MENU mbar:HANDLE
          CURRENT-WINDOW:VISIBLE = TRUE.
WAIT-FOR CHOOSE OF MENU-ITEM exit.
```

Notes

- You also can disable database triggers from the Data Dictionary.
- Triggers disabled with the DISABLE TRIGGERS statement remain disabled for the duration of the procedure in which you issued the statement and any subprocedures.
- The OpenEdge Data Dictionary automatically disables the appropriate triggers during data dump and load operations.

See also ON statement, TRIGGER PROCEDURE statement

DISCONNECT statement

Disconnects the specified database.

Syntax

```
DISCONNECT
  { logical-name | VALUE ( expression ) }
   [ NO-ERROR ]
```

logical-name

A logical database name. It can be an unquoted string or a quoted string. The *logical-name* is previously set, at startup or with a CONNECT statement, by using the Logical Database Name (-ld) parameter. If a logical name was not specified using the -ld parameter, then the physical database filename, without the .db suffix, is the default logical name.

VALUE (*expression*)

A character-string expression that evaluates to a logical database name.

NO-ERROR

Specifies that any errors that occur when you try to disconnect are suppressed. After the DISCONNECT statement completes, you can check the ERROR-STATUS system handle for information on errors that might occurred.

Example

This procedure disconnects the database with logical name mydb:

r-discnt.p

```
DISCONNECT mydb.
```

Notes

- By default, the Progress 4GL disconnects all databases at the end of a session. The DISCONNECT statement, which explicitly disconnects a database, does not execute until all active procedures that reference the database end or stop.
- If a transaction is active for *logical-name*, DISCONNECT is deferred until the transaction completes or is undone. If a CONNECT statement for the same *logical-name* database is executed before the same transaction completes or is undone, then the pending CONNECT and DISCONNECT cancel each other and the database remains connected.
- When the database referred to by *logical-name* is disconnected, existing aliases for *logical-name* remain in existence. Later, if you connect to a database with the same *logical-name*, the same alias is still available.

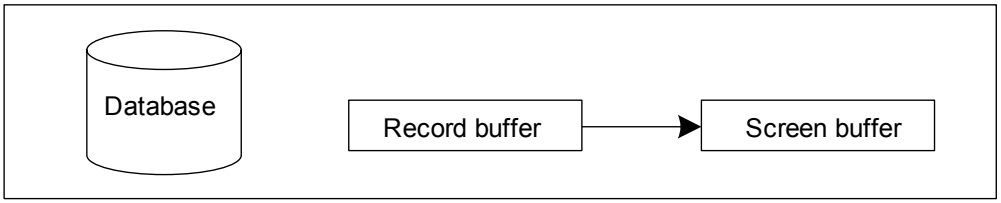
See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

DISPLAY statement

Moves data to a screen buffer and displays the data on the screen or other output destination. Progress uses frames to display data. A frame describes how constant and variable data is arranged for display and data entry. You can let Progress construct default frames or you can explicitly describe frames and their characteristics.

Data movement



Syntax

```
DISPLAY
  { [ STREAM stream ] [ UNLESS-HIDDEN ] }
  [    { expression
             [ format-phrase ]
             [ ( aggregate-phrase ) ]
             [ WHEN expression ]
             [ @base-field ]
       }
     |  [ SPACE [ ( n ) ] ]
     |  [ SKIP [ ( n ) ] ]
  ] ...
  { [ IN WINDOW window ] [ frame-phrase ] [ NO-ERROR ] }
```

```
DISPLAY
  { [ STREAM stream ] [ UNLESS-HIDDEN ] }
     record [ EXCEPT field ... ]
  { [ IN WINDOW window ] [ frame-phrase ] [ NO-ERROR ] }
```

```
DISPLAY
 {     expression ...
   |   record [ EXCEPT field ... ]
 }
  WITH BROWSE browse [ NO-ERROR ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

UNLESS-HIDDEN

Restricts DISPLAY to fields whose HIDDEN attribute is FALSE.

expression

Identifies a constant, field name, variable name, or expression that results in the value you want to display. This can also be the built-in field name, proc-text, that returns a character string of column values from a row returned by a stored procedure proc-text-buffer.

If *expression* is a simple field or variable, Progress checks to see if that particular field or variable is used previously in the same frame. If it has, Progress displays the field or variable in the same frame field as the earlier instance of that field or variable.

In array fields, array elements with constant subscripts are treated just as any other field. Array fields with no subscripts are expanded as though you had typed in the implicit elements.

Note: You cannot display elements of an indeterminate array parameter or variable. If you need to display one or more elements of an indeterminate array parameter or variable, assign those elements to a determinate array variable and display them from that variable.

If you reference `a[i]` in the same frame that you reference a or `a[constant]`, `a[i]` overlays the appropriate frame field based on the value of i. It is displayed in a new frame field for `a[i]`. For example, examine this procedure.

r-arry.p

```
1    DEFINE VARIABLE i AS INTEGER.

2    FOR EACH salesrep:
3       DISPLAY sales-rep region month-quota.
4       DO i = 1 TO 12:
5          SET month-quota[i] WITH 1 COLUMN.
6       END.
7       DISPLAY month-quota WITH FRAME a COLUMN 40 ROW 3 1 COLUMN.
8    END.
```

Here, `month-quota[i]` is referenced in the same frame that `month-quota` is referenced. That is, line 5 references `month-quota[i]` and line 3 references `month-quota`. Both references use the same frame. Therefore, instead of creating a new frame field for `month-quota[i]`, Progress uses the same frame fields created for the entire `month-quota` array.

In the next procedure, line 4 references only elements 1 and 2. Therefore, when Progress tries to overlay `month-quota[i]` in line 6, there is only room for elements 1 and 2. Progress returns an error after you enter data for those two elements.

r-arry2.p

```
1    DEFINE VARIABLE i AS INTEGER.

2    FOR EACH salesrep:
3      DISPLAY sales-rep region
4              month-quota[1] month-quota[2].
5      DO i = 1 TO 12:
6         SET month-quota[i] WITH 1 COLUMN.
7      END.
8      DISPLAY month-quota WITH FRAME a COLUMN 40 ROW 3 1 COLUMN.
9    END.
```

The following example shows a solution to that problem:

r-arry3.p

```
1    DEFINE VARIABLE i AS INTEGER.

2    FOR EACH salesrep:
3       DISPLAY sales-rep region
4               month-quota[1] month-quota[2] WITH 6 DOWN.
5       FORM i month-quota[i].
6       DO i = 1 TO 12:
7          DISPLAY i NO-LABEL.
8          SET month-quota[i].
9       END.
10       DISPLAY month-quota WITH FRAME a COLUMN 40 ROW 3 1 COLUMN.
11    END.
```

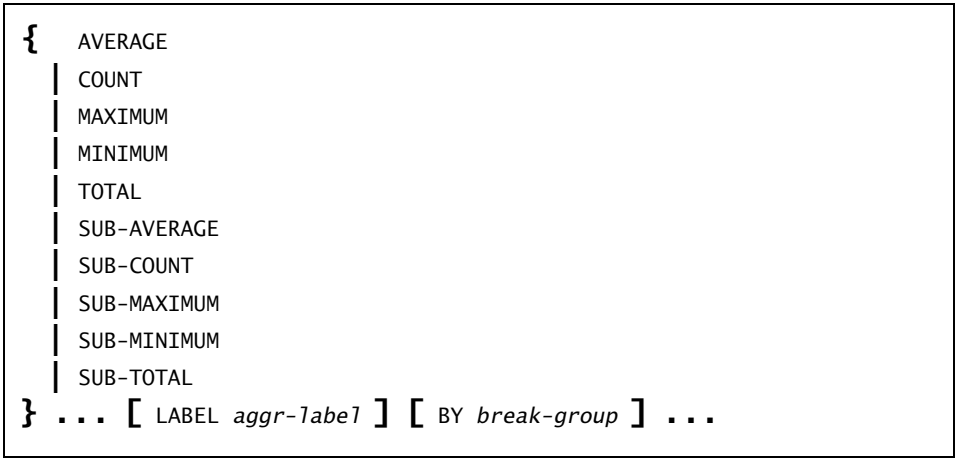
If you explicitly reference `a[i]` in a FORM statement, regular array fields (`month-quota[1]` and `month-quota[2]` in this example) are not overlaid.

format-phrase

Specifies one or more frame attributes for a field, variable, or expression. For more information on *format-phrase*, see the Format phrase reference entry.

aggregate-phrase

Identifies one or more aggregate values to be calculated optionally based on a change in a break group. This is the syntax for *aggregate-phrase*:



For more information on *aggregate-phrase*, see the Aggregate phrase reference entry.

WHEN *expression*

Displays an item only when the expression used in the WHEN option has a value of TRUE. Here, *expression* is a field name, variable name, or expression whose value is logical.

@ *base-field*

The *base-field* must be the name of a field or variable; it cannot be an expression or constant. The field or variable must be viewed as a fill-in or text widget on the display.

Progress reserves enough space for the *base-field* to hold the longest format displayed there. All right-justified fields (numerics that do not use side labels) are right justified within the reserved area. The label is left or right justified according to the *base-field*. Whenever you enter data into the *base-field* Progress blanks out any characters to the left or right of the area used by the field being displayed.

Progress underlines a screen area that is the longer of the *base-field* and the overlaying field. However, you can enter as many characters as there are spaces in the format of the field.

To determine the format to use for displaying the expression at the *base-field*, Progress looks at the following and uses the first format that applies:

- An explicit Format phrase used with the *expression*.
- If the expression is a character string constant, a format that accommodates that string.
- If the data type of the expression matches that of the *base-field*, the format of the *base-field*.
- The standard format of the expression as if it were displayed without a *base-field*.

SPACE **[** (*n*) **]**

Identifies the number (*n*) of blank spaces Progress inserts after the displayed expression displays. The *n* can be 0. If the number of spaces is more than the spaces left on the current line of the frame, Progress starts a new line and discards extra spaces. If you do not use this option or do not use *n*, Progress inserts one space between items in the frame.

SKIP **[** (*n*) **]**

Identifies the number (*n*) of blank lines Progress needs to insert after the expression is displayed. The *n* can be 0. If you do not use this option, Progress does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option but do not specify *n*, or if *n* is 0, Progress starts a new line unless it is already at the beginning of a new line.

IN WINDOW *window*

Identifies the window where the expression is displayed. The expression *window* must evaluate to the handle of a window.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

record

Identifies the name of the record you want to display. Naming a record is shorthand for listing each field individually. This can also be the built-in buffer name, proc-text-buffer, that returns each row retrieved by a stored procedure.

To display a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

EXCEPT *field* **. . .**

Indicates that Progress displays all fields except those fields listed in the EXCEPT phrase.

WITH BROWSE *browse*

Indicates that Progress displays the values into the current row of the specified browse widget.

Note: DISPLAY . . . WITH BROWSE cannot be used with a dynamic browse. Instead, the user must set the browse column's SCREEN-VALUE attributes.

NO-ERROR

Specifies that any errors that occur when you try to display the data are suppressed. After the DISPLAY statement completes, you can check the ERROR-STATUS system handle for information on any errors that might have occurred.

Examples

This procedure generates a hierarchical report of customers (sorted by state and name), the orders belonging to those customers, and the order-lines belonging to each order:

r-disp.p

```
FOR EACH customer BY state BY name:
     DISPLAY state cust-num name.
     FOR EACH order OF customer:
         DISPLAY order-num name ship-date promise-date.
         FOR EACH order-line OF order, item OF order-line:
             DISPLAY line-num item-name qty order-line.price.
         END.
     END.
END.
```

This procedure lists each order, customer information, and the order-lines for each order. The procedure calculates an Order-value for each of the order-lines of an order, and adds those values to produce a total value for an entire order.

r-disp2.p

```
FOR EACH order, customer OF order:
     DISPLAY order-num customer.name ship-date promise-date.
     FOR EACH order-line OF order, item OF order-line:
         DISPLAY line-num item-name qty order-line.price
                 qty * order-line.price (TOTAL) LABEL "Order-value".
     END.
END.
```

The `r-disp3.p` procedure displays a name and address list in a mailing label. The SKIP and FORMAT options are used to produce a standard address format. The WHEN option suppresses the display of the postal-code field if there is no postal code value in the field.

r-disp3.p

```
FOR EACH customer:
    DISPLAY name SKIP address SKIP address2 SKIP
            city + ", " + state FORMAT "x(16)"
            postal-code WHEN postal-code NE "" SKIP(2)
            WITH NO-BOX NO-LABELS USE-TEXT.
END.
```

Notes

- When Progress compiles a procedure, it uses a top-to-bottom pass of the procedure to design all the frames that procedure requires, adding field and related format attributes as it goes through the procedure.
- If you are displaying data that contains special control characters such as tabs, form feeds, or backspaces, be sure to use an EDITOR widget of the appropriate size for *expression* or *base-field*, or use the VIEW-AS EDITOR option from *format-phrase* in the DISPLAY statement. Otherwise, do not display data containing these characters.
- If you use a single qualified identifier with the DISPLAY statement, the Compiler first interprets the reference as *dbname.table-name*. If the Compiler cannot resolve the reference as *dbname.table-name*, it tries to resolve it as *table-name.fieldname*.

- If you invoke the DISPLAY statement for a frame, Progress brings the frame into view unless the HIDDEN attribute for the frame or one of its ancestor frames or windows is TRUE.
- For more information on using the built-in field and buffer names, proc-text and proc-text-buffer in a DISPLAY statement, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.
- You cannot use the DISPLAY statement to display an object reference for a class object instance directly. To display an object reference, you must first convert it using the INTEGER or STRING function and display the result.

See also ACCUM function, Aggregate phrase, DEFINE BROWSE statement, DEFINE FRAME statement, DOWN statement, EXPORT statement, FORM statement, Format phrase, Frame phrase, MESSAGE statement, PAGE statement, PUT statement, PUT SCREEN statement, UP statement, VIEW-AS phrase

DO statement

Groups statements into a single block, optionally specifying processing services or block properties. Use an END statement to end a DO block.

Syntax

```
[ label : ]
  DO
    { [ FOR record [ , record ] ... ] }
    [ preselect-phrase ]
    [ query-tuning-phrase ]
    [ variable = expression1 TO expression2 [ BY k ] ]
    [ WHILE expression ]
    [ TRANSACTION ]
    [ on-endkey-phrase ]
    [ on-error-phrase ]
    [ on-quit-phrase ]
    [ on-stop-phrase ]
    { [ frame-phrase ] }
```

`FOR` *record* **[** , *record* **]** **...**

Names the buffer you want to work with in the block and scopes the buffer to the block. The scope of a record determines when the buffer for that record is cleared and written back to the database. See *OpenEdge Development: Progress 4GL Handbook* for more information on record scoping.

To work with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

preselect-phrase

The PRESELECT phrase finds selected records from one or more tables. You can access those preselected records with statements such as FIND NEXT.

```
PRESELECT
  [ EACH | FIRST | LAST ] record-phrase
  [ , [ EACH | FIRST | LAST ] record-phrase ] ...
  [
     [ BREAK ]
     { BY expression [ DESCENDING ] } ...
  ]
```

For more information, see the PRESELECT phrase reference entry.

query-tuning-phrase

Allows programmatic control over the execution of a DataServer query.

```
QUERY-TUNING (
  {
     [ BIND-WHERE | NO-BIND-WHERE ]
     [ CACHE-SIZE integer ]
     [ DEBUG { SQL | EXTENDED } | NO-DEBUG ]
     [ INDEX-HINT | NO-INDEX-HINT ]
     [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
     [ LOOKAHEAD | NO-LOOKAHEAD ]
     [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
  }
  )
```

For more information, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

variable = *expression1* TO *expression2* [BY *k*]

The name of a field or variable whose value is incremented in a loop. The *expression1* is the starting value for *variable* on the first iteration of the loop. The *k* is the amount to add to *variable* after each iteration, and it must be a constant. The *k* defaults to 1. The *variable*, *expression1* and *expression2* must be integers.

When *variable* exceeds *expression2* (or is less than *expression2* if *k* is negative) the loop ends. Since *expression1* is compared to *expression2* at the start of the first iteration of the block, the block can be executed zero times. The *expression2* is re-evaluated on each iteration of the block.

WHILE *expression*

Indicates that the DO block continues processing the statements within it. Using the WHILE option turns a DO block into an iterating block. The block iterates as long as the condition specified by the expression is TRUE. The expression is any combination of constants, operators, field names, and variable names that yield a logical value.

TRANSACTION

Identifies the DO block as a system transaction block. Progress starts a system transaction for each iteration of a transaction block if there is not already an active system transaction. See *OpenEdge Development: Progress 4GL Handbook* for more information on transactions.

on-endkey-phrase

Describes the processing that takes place when the ENDKEY condition occurs during a block. This is the syntax for a ON ENDKEY phrase:

```
ON ENDKEY UNDO [ label1 ] [ , LEAVE label2 ]
```

```
ON ENDKEY UNDO [ label1 ] [ , NEXT label2 ]
```

```
ON ENDKEY UNDO [ label1 ] [ , RETRY label1 ]
```

```
ON ENDKEY UNDO [ label1 ]
  [
    , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON ENDKEY phrase reference entry.

on-error-phrase

Describes the processing that takes place when there is an error during a block. This is the syntax for ON ERROR phrase:

```
ON ERROR UNDO [ label1 ] [ , LEAVE label2 ]
```

```
ON ERROR UNDO [ label1 ] [ , NEXT label2 ]
```

```
ON ERROR UNDO [ label1 ] [ , RETRY label1 ]
```

```
ON ERROR UNDO [ label1 ]
  [
    , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON ERROR phrase reference entry.

on-quit-phrase

Describes the processing that takes place when a QUIT statement is executed during a block. This is the syntax for ON QUIT phrase:

```
ON QUIT UNDO [ label1 ] [ , LEAVE label2 ]
```

```
ON QUIT UNDO [ label1 ] [ , NEXT label2 ]
```

```
ON QUIT UNDO [ label1 ] [ , RETRY label1 ]
```

```
ON QUIT UNDO [ label1 ]
  [
    , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON QUIT phrase reference entry.

on-stop-phrase

Describes the processing that takes place when the STOP conditions occurs during a block. This is the syntax for the ON STOP phrase:

```
ON STOP UNDO [ label1 ] [ , LEAVE label2 ]
```

```
ON STOP UNDO [ label1 ] [ , NEXT label2 ]
```

```
ON STOP UNDO [ label1 ] [ , RETRY label1 ]
```

```
ON STOP UNDO [ label1 ]
  [
    , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON STOP phrase reference entry.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

This procedure goes through the customer table and, for those customers whose credit-limit is over 80000, reduces credit-limit to 80000. The procedure uses an unmodified DO block to process two statements if credit-limit is over 80000. Unmodified DO blocks are most useful in conditional, or IF. . .THEN. . .ELSE situations.

r-do.p

```
FOR EACH customer:
  DISPLAY name credit-limit.
  PAUSE 3.
  IF credit-limit > 80000 THEN DO:
       credit-limit = 80000.
       DISPLAY name credit-limit.
  END.
END.
```

Notes

- Use a DO statement rather than a REPEAT statement when you loop through each element of an array. This way Progress does not create separate subtransactions within a transaction.

  For example, the first transaction is more efficient then the second:

```
DO i = 1 TO 12:
  month-quota[i] = 0.
END.
```

```
REPEAT i = 1 TO 12:
  month-quota[i] = 0.
END.
```

- For SpeedScript, the *on-endkey-phrase* and the *on-quit-phrase* do not apply.

See also

FIND statement, FOR statement, Frame phrase, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, ON STOP phrase, Record phrase, REPEAT statement

DOS statement (Windows only)

Runs a program, DOS command, or DOS batch file, or starts the DOS command processor, which allows interactive processing of DOS commands.

Syntax

```
DOS [ SILENT ] [ command-token | VALUE ( expression ) ] ...
```

SILENT

After processing a DOS statement, the Progress shell pauses, and prompts you to press **SPACEBAR** to continue. When you press **SPACEBAR**, Progress clears the window and continues processing. You can use the SILENT option to eliminate this pause. Use this option only if you are sure that the DOS program, command, or batch file will not generate output to the window.

command-token | VALUE (*expression*)

One or more command (*command-token*) words and symbols that you want to pass to a DOS command processor. The VALUE option generates the command tokens included in *expression*, a character string expression. The specified combination of *command-token* and VALUE (*expression*) options can form any legal combination of commands and command options permitted by the DOS command processor, including programs, DOS commands, and batch files. If you do not use any of these options, the DOS statement invokes the DOS command processor, which remains until you exit it.

Example

On UNIX, this procedure runs the UNIX `ls` command. In Windows, this procedure runs the DOS `dir` command. On other platforms, Progress displays a message stating that the operating system is unsupported.

r-dos.p

```
IF OPSYS = "UNIX" THEN UNIX ls.
ELSE IF OPSYS = "WIN32" THEN DOS dir.
ELSE DISPLAY OPSYS "is an unsupported operating system".
```

Note

If you use the DOS statement in a procedure and the procedure compiles on a UNIX system, the procedure runs, as long as flow of control does not pass through the DOS statement while running on UNIX. Use the OPSYS function to return the name of the operating system where a procedure is being run. This function lets you write applications that are portable among Progress-supported operating systems even if they use the DOS, UNIX, etc. statements.

See also

{ } Preprocessor name reference, OPSYS function, UNIX statement

DOWN statement

Positions the cursor on a new line in a down or multi-line frame.

When the block specifying the down frame iterates, Progress automatically advances one frame line. Use the DOWN statement if you want to move to a different display line at any time.

For more information on down frames, see the DOWN option of the Frame phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
DOWN [ STREAM stream ] [ expression ] { [ frame-phrase ] }
```

`STREAM stream`

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry and the "Alternate I/O Sources" chapter in *OpenEdge Development: Programming Interfaces* for more information on streams.

expression

The number of occurrences of data in the frame that you want to move down.

DOWN is the same as DOWN 1, except for the following:

- Nothing happens until a data handling statement affects the screen.
- Several DOWN statements in a row with no intervening displays are treated as a single DOWN 1.

DOWN 0 does nothing. If *n* is negative, the result is the same as UP ABS(*n*).

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

This procedure prints a customer report that is sorted by state, with one line after the last customer in each state:

r-down.p

```
DEFINE VARIABLE laststate AS CHARACTER.

FOR EACH customer BY state:
    IF state <> laststate THEN DO:
         IF laststate <> "" THEN DOWN 1.
         laststate = state.
    END.
    DISPLAY cust-num name city state.
END.
```

Notes

- After displaying a down frame, Progress automatically advances to the next frame line on each iteration of the block where the frame belongs. This is true whether or not you use the DOWN statement. If you do not want Progress to advance automatically, name the frame outside of the block involved (the statement FORM WITH FRAME *frame* names a frame and scopes that frame to the higher block).
- When Progress reaches the last frame line and encounters a DOWN statement, it clears the frame and starts at the top line of the frame, unless you used the SCROLL option on the frame. In that case, Progress scrolls the frame up one iteration only, to make room for the next iteration.

See also

DEFINE STREAM statement, Frame phrase, SCROLL statement, UP statement

DYNAMIC-CURRENT-VALUE function

Returns the current integer value of a sequence defined in the specified database.

Syntax

```
DYNAMIC-CURRENT-VALUE( sequence-expression, logical-dbname-expression )
```

sequence-expression

A character expression that evaluates to the name of a sequence.

logical-dbname-expression

A character expression that evaluates to the name of a connected database in which the sequence is defined.

Notes

- If *logical-dbname-expression* contains the Unknown value (?), Progress generates a runtime error.
- The user must have CAN-READ privileges on the _Sequence table to use the DYNAMIC-CURRENT-VALUE function.
- The current value of a sequence can be one of the following:
  - The initial value specified in the Data Dictionary.
  - The last value set with either the DYNAMIC-CURRENT-VALUE statement or the DYNAMIC-NEXT-VALUE function.
  - The Unknown value (?) if the sequence has exceeded its minimum or maximum and is not cycling.
- Sequence values are stored in the database in which they are defined, and persist between each invocation of the DYNAMIC-CURRENT-VALUE statement or DYNAMIC-NEXT-VALUE function.

- You cannot invoke the DYNAMIC-CURRENT-VALUE function from within a WHERE clause. Doing so generates a compiler error. To use a result from the DYNAMIC-CURRENT-VALUE function in a WHERE clause, assign the result to a variable, then use the variable in the WHERE clause.
- You can use any combination of the DYNAMIC-NEXT-VALUE function, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, and their static versions.

See also CURRENT-VALUE function, CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

DYNAMIC-CURRENT-VALUE statement

Resets the current integer value of a sequence defined in the specified database.

Syntax

```
DYNAMIC-CURRENT-VALUE( sequence-exp, logical-dbname-exp ) = expression
```

sequence-exp

A character expression that evaluates to the name of a sequence.

logical-dbname-exp

A character expression that evaluates to the name of a connected database in which the sequence is defined.

expression

An integer expression assigned as the current value of the specified sequence. If *expression* is outside the boundary set by the initial value (at one end) and the lower limit or upper limit (at the other end) for the sequence, Progress returns an error, and the sequence value remains unchanged.

Notes

- If *logical-dbname-exp* contains the Unknown value (?), Progress generates a runtime error.
- You cannot set a sequence to the Unknown value (?).
- The user must have CAN-WRITE privileges on the _Sequence table to use the DYNAMIC-CURRENT-VALUE statement.
- The value of a sequence set by the DYNAMIC-CURRENT-VALUE statement persists in the database until the next DYNAMIC-CURRENT-VALUE statement or DYNAMIC-NEXT-VALUE function is invoked for the sequence, or until the sequence is deleted from the database.
- You can use any combination of the DYNAMIC-NEXT-VALUE function, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, and their static versions.

See also

CURRENT-VALUE function, CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

DYNAMIC-FUNCTION function

Invokes a user-defined function. Progress evaluates the name of the function (and the procedure handle, if any) at run time.

Syntax

```
DYNAMIC-FUNCTION
  ( function-name [ IN proc-handle ]
    [ , param1 [ , param2 ] ... ]
  )
```

function-name

A CHARACTER expression that returns the name of a user-defined function. Progress evaluates *function-name* at run time.

IN *proc-handle*

An expression that returns a handle to the procedure that defines the function. Progress evaluates *proc-handle* at run time.

param1, param2, ...

Parameters of the user-defined function. You must supply names of actual data items—actual parameter names—not CHARACTER expressions that return parameter names.

Note: Progress cannot check the mode and type of the parameters at compile time, since Progress does not evaluate *function-name* until run time.

Example The following procedure demonstrates the DYNAMIC-FUNCTION function:

r-funfun.p *(1 of 2)*

```
/* Requires a connection to the Sports database *//* define data items */DEFINE
VAR funcs AS Char EXTENT 5 INITIAL ["firstrec",
                                    "lastrec",
                                    "nextrec",
                                    "prevrec",
                                    "quitting"] NO-UNDO.
DEFINE VAR action AS Char LABEL "Action" FORMAT "x" INITIAL "N" NO-UNDO.
DEFINE VAR idx AS Int NO-UNDO.
DEFINE VAR alldone AS Logical INITIAL No NO-UNDO.FORM WITH FRAME x SIDE-LABELS
2 COLUMNS 1 DOWN COLUMN 25.FUNCTION dispcust RETURNS Logical:
  DISPLAY Customer EXCEPT Comments WITH FRAME x.
END.
/* define user-defined functions */FUNCTION firstrec RETURNS Logical:
  FIND FIRST Customer.
  dispcust().
  RETURN yes.
END.
FUNCTION lastrec RETURNS Logical:
  FIND LAST Customer.
  dispcust().
  RETURN yes.
END.

/* define more user-defined functions */
FUNCTION nextrec RETURNS Logical:
  FIND NEXT Customer NO-ERROR.
  IF AVAILABLE Customer THEN
     dispcust().
  RETURN AVAILABLE(Customer).
END.
```

r-funfun.p *(2 of 2)*

```
FUNCTION prevrec RETURNS Logical:
  FIND PREV Customer NO-ERROR.
  IF AVAILABLE Customer THEN
     dispcust().
  RETURN AVAILABLE(Customer).
END.

FUNCTION quitting RETURNS Logical:
  alldone = yes.
  RETURN no.
END

/* main routine */
REPEAT WHILE NOT alldone:
  UPDATE action HELP
    "Enter F(irst), L(ast), N(ext), P(rior), or Q(uit) to navigate.".
  idx = LOOKUP(action,"f,l,n,p,q").
  IF idx EQ 0 THEN DO:
    MESSAGE "Enter F(irst), L(ast), N(ext), P(rior), or Q(uit)"
        VIEW-AS ALERT-BOX.
    NEXT.
  END.
  DISPLAY DYNAMIC-FUNCTION(funcs[idx]) LABEL "Record Found?".
END.
```

See also FUNCTION statement

DYNAMIC-NEXT-VALUE function

Returns the next integer value of a sequence, incremented by the positive or negative value defined in the specified database.

Syntax

```
DYNAMIC-NEXT-VALUE( sequence-expression, logical-dbname-expression )
```

sequence-expression

A character expression that evaluates to the name of a sequence.

logical-dbname-expression

A character expression that evaluates to the name of a connected database in which the sequence is defined.

Notes

- If *logical-dbname-expression* contains the Unknown value (?), Progress generates a runtime error.
- If *sequence-expression* is a cycling sequence, and the DYNAMIC-NEXT-VALUE function increments the sequence beyond its upper limit (for positive increments) or decrements the sequence beyond its lower limit (for negative increments), the function sets and returns the initial value defined for the sequence.
- If *sequence-expression* is a terminating sequence, and the DYNAMIC-NEXT-VALUE function attempts to increment the sequence beyond its upper limit (for positive increments) or decrement the sequence beyond its lower limit (for negative increments), the function returns the Unknown value (?) and leaves the current sequence value unchanged. Once a sequence terminates, DYNAMIC-NEXT-VALUE continues to return the Unknown value (?) for the specified sequence until it is reset to a new value with the DYNAMIC-CURRENT-VALUE statement, or its definition is changed to a cycling sequence. After changing the sequence definition to cycle, the first use of DYNAMIC-NEXT-VALUE for the sequence sets and returns its initial value.

- The value of a sequence set by the DYNAMIC-NEXT-VALUE function persists in the database until the next DYNAMIC-CURRENT-VALUE statement or DYNAMIC-NEXT-VALUE function is invoked for the sequence, or until the sequence is deleted from the database.
- You cannot invoke the DYNAMIC-NEXT-VALUE function from within a WHERE clause. Doing so generates a compiler error because the value returned by the DYNAMIC-NEXT-VALUE function can result in ambiguous expressions. To use a result from the DYNAMIC-NEXT-VALUE function in a WHERE clause, assign the result to a variable and use the variable in the WHERE clause instead.
- You can use any combination of the DYNAMIC-NEXT-VALUE function, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, and their static versions.

See also CURRENT-VALUE function, CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, NEXT-VALUE function

EDITING phrase

Identifies the process that follows each keystroke during a PROMPT-FOR, SET, or UPDATE statement.

This phrase is maintained primarily for compatibility with Progress Version 6 or earlier.

Note: Does not apply to SpeedScript programming.

Syntax

```
[ label: ] EDITING: statement ... END
```

statement

One or more statements you want to process, usually for each keystroke entered. In most cases, the first statement is READKEY.

Example

This procedure lets you update the i variable, and immediately processes each of your keystrokes. The READKEY statement reads each of the keys you press. The APPLY statement applies, or executes, each keystroke. This is a very simple EDITING phrase and is the same as entering UPDATE i.

r-edit.p

```
DEFINE VARIABLE i AS INTEGER.

UPDATE i EDITING:
  READKEY.
  APPLY LASTKEY.
END.
```

The following r-edit2.p procedure uses an EDITING phrase with an UPDATE statement to control what happens based on each keystroke during the UPDATE. Here, the user can press any key while updating any field except sales-rep.

While in the sales-rep field, the user can press **SPACEBAR** to scroll through the possible values for the sales-rep field. If the user presses the **TAB**, **BACKTAB**, **GO**, **RETURN**, or **END-ERROR** key, the procedure executes that key. If the user presses any other key while in the sales-rep field, the terminal beeps.

r-edit2.p

```
PROMPT-FOR customer.cust-num.
FIND customer USING cust-num.

/* Update customer fields, monitoring each keystroke during the UPDATE */
UPDATE name address city state SKIP
       sales-rep HELP "Use the space bar to select a sales-rep"
       WITH 2 COLUMNS
    EDITING:  /* Read a keystroke */
      READKEY.
      /* If the cursor is in any field except sales-rep, execute the
         last key pressed and go on to the next iteration of this
         EDITING phrase to check the next key */
     IF FRAME-FIELD <> "sales-rep" THEN DO:
        APPLY LASTKEY.
        IF GO-PENDING THEN LEAVE.
        ELSE NEXT.
     END.
     /* When in the sales-rep field, if the last key pressed was
        the space bar then cycle through the sales reps */
     IF LASTKEY = KEYCODE(" ") THEN DO:
        FIND NEXT salesrep NO-ERROR.
        IF NOT AVAILABLE salesrep THEN FIND FIRST salesrep.
        DISPLAY salesrep.sales-rep @ customer.sales-rep.
        NEXT.
     END.
     /* If the user presses any one of a set of keys while in the
        sales-rep field, immediately execute that key */
     IF LOOKUP(KEYFUNCTION(LASTKEY),
               "TAB,BACK-TAB,GO,RETURN,END-ERROR") > 0
     THEN APPLY LASTKEY.
     ELSE BELL.
END.
```

Notes

- A READKEY statement does not have to be the first statement after the word EDITING. However, it should appear somewhere in the EDITING phrase because Progress does not automatically read keystrokes when you use an EDITING phrase.
- The EDITING phrase applies to the PROMPT-FOR part of a SET or UPDATE statement. Therefore, to examine a value supplied by the user (within an EDITING phrase), you must use the INPUT function to refer to the field or variable that contains the value.

- When you use the NEXT statement in an EDITING phrase, Progress executes the next iteration of that EDITING phrase and cancels any pending GO.
- When you use the LEAVE statement in an EDITING phrase, Progress leaves the EDITING phrase and executes the assignment part of the SET or UPDATE statement.
- Within an EDITING phrase, you cannot use the CLEAR ALL, DOWN, or UP statements on the frame being edited.
- If you hide and redisplay a frame while you are in an EDITING block, Progress might not redisplay it in the same location unless you specifically name the row and column of the frame. This could cause problems because the EDITING block does not recognize the new location, and attempts to update the fields at the old frame location.
- The EDITING phrase activates only for input from a terminal. If your input comes from an operating system file (set with the INPUT FROM statement), the EDITING phrase has no effect.
- The EDITING phrase is incompatible with event-driven programming. An EDITING block might interfere with other event handling statements.
- For more information on EDITING blocks and other ways of monitoring keystrokes, see *OpenEdge Development: Programming Interfaces*.

See also END statement, PROMPT-FOR statement, READKEY statement, SET statement, UPDATE statement

EDITOR phrase

Specifies that a field or variable is displayed as a text editor widget. This is especially useful for long text (CHARACTER and LONGCHAR) fields. The EDITOR phrase is an option of the VIEW-AS phrase.

Syntax

```
EDITOR
  { size-phrase
    | INNER-CHARS characters INNER-LINES lines
  }
  [ BUFFER-CHARS chars ]
  [ BUFFER-LINES lines ]
  [ LARGE ]
  [ MAX-CHARS characters ]
  [ NO-BOX ]
  [ NO-WORD-WRAP ]
  [ SCROLLBAR-HORIZONTAL ]
  [ SCROLLBAR-VERTICAL ]
  [ TOOLTIP tooltip ]
```

size-phrase

Specifies the outer width and height of the text editor widget in characters or pixels. This is the syntax for size-phrase:

Syntax

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information, see the SIZE phrase reference entry.

INNER-CHARS *chars* INNER-LINES *lines*

Specifies the number of characters visible in each line of the Editor and the number of lines visible within the Editor. Both *chars* and *lines* must be integer constants.

Note that the values you supply for INNER-CHARS and INNER-LINES specify only the size of the editing area, not the overall size of the editor widget. The overall size is determined by the size of the editing area plus the sizes of the margin and border heights and widths.

BUFFER-CHARS *chars*

In character mode, specifies the number of characters a user can enter on each line. When the last character is typed, the text input cursor automatically wraps to the next line. This option is ignored in graphical environments.

The *chars* value must be an integer constant that is equal to or greater than the value specified by SIZE *width* or INNER-CHARS *chars*. If greater, horizontal scrolling is enabled. The default is the value specified by SIZE *width* or INNER-CHARS *chars*.

BUFFER-LINES *lines*

In character mode, specifies the number of lines a user can enter. By default, Progress does not limit the number of lines (although system limits might apply). This option is ignored in graphical environments.

The *lines* value must be an integer constant that is equal to or greater than the value specified by BY *height* or INNER-LINES *lines*. If equal, vertical scrolling is disabled.

LARGE

Specifies that Progress use a large editor widget rather than a normal editor widget in Windows. A normal Windows editor can contain up to 20K of data. The LARGE option allows the editor to contain data up to the limit of your system resources. However, it also consumes more internal resources and lacks some functionality. Use the LARGE option only if you have to edit very large sections of text. The LARGE option applies only to Windows; other interfaces allow for larger editors by default. This option is ignored in those other interfaces.

MAX-CHARS *characters*

The maximum number of characters that can be displayed or entered within the text editor widget. The value *characters* must be an integer constant. By default, Progress does not limit the number of characters (although system limits might apply).

NO-BOX

Specifies that the editor be displayed without a border. The default is to display the editor with a border. The NO-BOX option has no effect on the size of the editor.

NO-WORD-WRAP

Specifies that word wrap be disabled within the text editor widget. If you enable word wrap, horizontal scrolling is disabled. This option is ignored in character mode. This is the default with the LARGE option.

SCROLLBAR-HORIZONTAL

Specifies that horizontal scrolling is enabled and a horizontal scroll bar is displayed for the widget.

SCROLLBAR-VERTICAL

Specifies that a vertical scroll bar is display for the widget. Although vertical scrolling is always enabled within a text editor widget, a vertical scroll bar is displayed only if you specify this option.

TOOLTIP *tooltip*

Allows you to define a help text message for a text field or text variable. Progress automatically displays this text when the user pauses the mouse button over a text field or text variable for which a tooltip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the tooltip is removed. No tooltip is the default. The TOOLTIP option is supported in Windows only.

Example

The following example uses two editor widgets. The item.cat-description field is viewed as an editor in the item-info frame and the variable my_clipbd is viewed as an editor in the clip frame. Use the editor functions provided by your interface environment to copy text from a cat-description into my_clipbd. You can then subsequently copy that text into the cat-description field of another item.

r-vaedit.p

```
DEFINE VARIABLE my_clipbd AS CHARACTER VIEW-AS EDITOR SIZE 60 BY 6
                              SCROLLBAR-VERTICAL LABEL "Scratch Pad".

DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.

FORM
  item.item-num
  item.item-name
  item.price
  item.on-hand
  item.allocated
  item.re-order
  item.on-order
  item.cat-page
  item.cat-description VIEW-AS EDITOR SIZE 35 BY 3 SCROLLBAR-VERTICAL
  WITH FRAME item-info 1 DOWN ROW 1 CENTERED SIDE-LABELS
  TITLE "Update Item Category Description".

FORM
  my_clipbd
  WITH FRAME clip.

DEFINE FRAME butt-frame
  b_quit
WITH CENTERED ROW 18.

ON GO OF item.item-num
  DO:
    FIND item USING item.item-num EXCLUSIVE-LOCK.
    DISPLAY item WITH FRAME item-info.
    ENABLE item.cat-description WITH FRAME item-info.
    ENABLE my_clipbd WITH FRAME clip.
  END.

ON GO OF item.cat-description
   DO:
     ASSIGN item.cat-description.
     CLEAR FRAME item-info.
     DISABLE item.cat-description WITH FRAME item-info.
   END.

ENABLE item.item-num WITH FRAME item-info.
ENABLE b_quit WITH FRAME butt-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- If you specify the SCROLLBAR-VERTICAL option in, a vertical scroll bar appears on the side of the Editor. The user can then use the scroll bar to scroll within the widget. Whether or not you specify SCROLLBAR-VERTICAL, the user can scroll vertically by using the up and down arrow keys to move above or below the displayed text.
- If you use the SIZE phrase to specify the dimensions of the Editor, Progress uses a portion of this overall space (thereby shrinking the size of the editing area) for any scroll bars you specify. Use the INNER-CHARS and INNER-LINES options if you want a fixed size for the editing area, regardless of the presence of scroll bars.
- In Windows, the editor widget supports lines of up to 255 characters only.
- By default, the editor widget supports text wrap. This means that when you reach the end of a line within the widget, text wraps to the next line rather than scrolling to the right. In graphical interfaces, you can enable horizontal scrolling by specifying either the NO-WORD-WRAP or SCROLLBAR-HORIZONTAL options. If you specify SCROLLBAR-HORIZONTAL, a horizontal scroll bar appears. If you specify NO-WORD-WRAP, but not SCROLLBAR-HORIZONTAL, the user can scroll horizontally by using the left and right arrow keys at the edge of the displayed text.
- Windows allows a user to transfer focus to the editor by pressing **ALT** and one of the letters in the label. This is called a *mnemonic*.
- The character-mode editor does not support the tab character. When Progress reads a file that contains tabs into an editor widget, it replaces the tabs with eight spaces. When it writes the file, the tabs are not restored and the file is permanently changed.

- When you specify the LARGE option, the following attributes and methods no longer apply to the editor:
  - CONVERT-TO-OFFSET method
  - CURSOR-OFFSET attribute
  - LENGTH attribute
  - MAX-CHARS attribute
  - SELECTION-END attribute
  - SELECTION-START attribute
  - SET-SELECTION method
  - WORD-WRAP attribute
- For SpeedScript, the only valid options are: *size-phrase*, INNER-CHARS, INNER-LINES, MAX-CHARS, NO-BOX, NO-WORD-WRAP.

See also SIZE phrase, VIEW-AS phrase

EMPTY TEMP-TABLE statement

Empties a temp-table.

Temp-tables defined as NO-UNDO can be emptied within a transaction. Temp-tables that are defined as UNDO or default to UNDO can only be emptied as long as no transaction is active, whether or not it involves records in the temp-table.

Note: This statement corresponds to the EMPTY-TEMP-TABLE( ) method.

Syntax

```
EMPTY TEMP-TABLE temp-table-name [ NO-ERROR ]
```

temp-table-name

The name of the temp-table.

NO-ERROR

Specifies that any errors that occur in the attempt to empty the temp-table are suppressed. After the EMPTY TEMP-TABLE statement completes, you can check the ERROR-STATUS system handle for any errors that occurred.

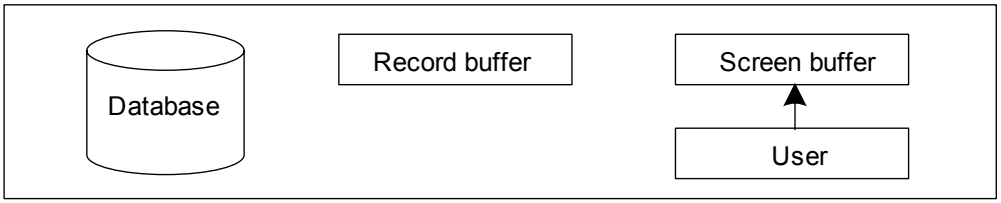
See also

EMPTY-TEMP-TABLE( ) method

ENABLE statement

Enables input for one or more field-level and child frame widgets within a frame.

Data movement



Syntax

```
ENABLE [ UNLESS-HIDDEN ]
  {    ALL [ EXCEPT field ... ]
     | {   field [ format-phrase ] [ WHEN expression ]
         | TEXT ( { field [ format-phrase ]
             [ WHEN expression ] } ... )
         | constant [ AT n | TO n ]
             [ BGCOLOR expression ]
             [ DCOLOR expression ]
             [ FGCOLOR expression ]
             [ FONT expression ]
             [ PFCOLOR expression ]
             [ VIEW-AS TEXT ]
         | SPACE [ ( n ) ]
         | SKIP [ ( n ) ]
       } ...
   }
  [ IN WINDOW window ] [ frame-phrase ]
```

`ALL [ EXCEPT field ...]`

Specifies that all field-level widgets for a frame are enabled, except those you list.

`UNLESS-HIDDEN`

Restricts ENABLE to fields whose HIDDEN attribute is FALSE.

field

Specifies the name of the field, variable, or widget you want to enable. Remember that the ENABLE statement accepts input only and stores it in the screen buffer. The underlying record buffer of a field or variable is unaffected unless you ASSIGN the value.

In array fields, array elements with constant subscripts are treated just like any other field. Array fields with no subscripts or array fields in the FORM statement are expanded as though you had entered the implicit elements. See the DISPLAY statement reference entry for information on how array fields with expressions as subscripts are handled.

Note: You cannot enable indeterminate array variables.

format-phrase

Specifies one or more frame attributes for a field, variable, or expression. For more information on *format-phrase*, see the Format phrase reference entry.

WHEN *expression*

Enables the field only if *expression* has a value of TRUE when the ENABLE statement is executed. Here, *expression* is a field name, variable name, or expression that evaluates to a LOGICAL value.

TEXT

Defines a group of character fields or variables (including array elements) to use automatic text-wrap. The TEXT option works only with character fields. When you insert data in the middle of a TEXT field, Progress wraps data that follows into the next TEXT field, if necessary. If you delete data from the middle of a TEXT field, Progress wraps data that follows into the empty area.

If you enter more characters than the format for the field allows, Progress discards the extra characters. The character fields must have formats in the form x(n). A blank in the first column of a line marks the beginning of a paragraph. Lines within a paragraph are treated as a group and will not wrap into other paragraphs.

constant [AT *n* | TO *n*] [BGCOLOR *expression*] [DCOLOR *expression*] [FGCOLOR *expression*] [FONT *expression*] [PFCOLOR *expression*] [VIEW-AS TEXT]

Specifies a constant (literal) value that you want displayed in the frame. If you use the AT option, *n* is the column in which you want to start the display. If you use the TO option, *n* is the column in which you want to end the display. You can use the BGCOLOR and FGCOLOR options in graphical interfaces to define the foreground and background colors to use when the constant is displayed. Similarly, you can use the DCOLOR and PFCOLOR options in character interfaces to define the prompt and display colors to use when the constant is displayed. The font option, for both character and graphical interfaces, defines the font used. If you use the VIEW-AS TEXT option, the constant is displayed as a text widget rather than a fill-in field.

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces to insert after the field displays. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, Progress starts a new line and discards any extra spaces. If you do not use this option or *n*, Progress inserts one space between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines to insert after the field is displays. The *n* can be 0. If you do not use this option, Progress does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option, but do not specify *n*, or if *n* is 0, Progress starts a new line unless it is already at the beginning of a new line.

^

Tells Progress to ignore an input field when input is being read from a file.

IN WINDOW *window*

Specifies the window in which the widgets are enabled. The *window* parameter must be the name of a currently defined window or an expression that evaluates to the handle for a currently defined window.

frame-phrase

The frame that contains the widgets to enable. If you omit *frame-phrase*, the default frame for the current block is assumed. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

The following example enables the cust-num field and the Quit button in the main procedure. If you press **GO** in the cust-num field and successfully find a record, the trigger disables the cust-num field and enables the credit-limit field and the Save and Undo buttons. If you choose Save or Undo, the CHOOSE trigger disables the buttons and enables the cust-num field again. Note that if you choose the Save button, the trigger must execute an ASSIGN statement to set the value in the underlying database field.

r-enable.p *(1 of 2)*

```
DEFINE VARIABLE ok AS LOGICAL NO-UNDO.

DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.
DEFINE BUTTON b_save LABEL "Save".
DEFINE BUTTON b_undo LABEL "Undo".

DEFINE FRAME butt-frame
  b_save b_undo b_quittt
  WITH CENTERED ROW SCREEN-LINES - 2.

FORM
  customer
WITH FRAME cust-info SIDE-LABELS CENTERED
    TITLE "Update Customer Credit Limit".

ON CHOOSE OF b_save, b_undo IN FRAME butt-frame
DO:
  DISABLE b_save b_undo WITH FRAME butt-frame.
  DISABLE customer.credit-limit WITH FRAME cust-info.
  ENABLE customer.cust-num WITH FRAME cust-info.
  IF SELF:LABEL = "save" THEN
    ASSIGN FRAME cust-info customer.credit-limit.
  CLEAR FRAME cust-info NO-PAUSE.
  APPLY "ENTRY" TO customer.cust-num IN FRAME cust-info.
END.
```

r-enable.p *(2 of 2)*

```
ON GO OF customer.cust-num IN FRAME cust-info
DO:
  FIND customer USING customer.cust-num EXCLUSIVE NO-ERROR.
  IF AVAILABLE(customer) THEN
  DO:
    DISABLE customer.cust-num WITH FRAME cust-info.
    ENABLE customer.credit-limit WITH FRAME cust-info.
    ENABLE ALL WITH FRAME butt-frame.
    DISPLAY customer WITH FRAME cust-info.
  END.
  ELSE
  DO:
     MESSAGE "No Customer Record exist for customer number"
             INPUT customer.cust-num ", Please re-enter."
       VIEW-AS ALERT-BOX WARNING BUTTONS OK-CANCEL UPDATE OK.
    IF NOT ok THEN
      APPLY "CHOOSE" TO b_quit IN FRAME butt-frame.
  END.
END.

ENABLE customer.cust-num WITH FRAME cust-info.
ENABLE b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame
   FOCUS customer.cust-num IN FRAME cust-info.
```

Notes

- For field representation widgets, the ENABLE statement lets you change the widget's SCREEN-VALUE. If you want to save changes to the field itself, you must subsequently use the ASSIGN statement.

  During data entry, a validation expression defined for the field in the database or in a Format phrase executes only if the widget associated with the field receives input focus. Use the VALIDATE( ) method to execute a validation expression defined for a field regardless of whether it receives input focus or not.

- If you invoke the ENABLE statement for a frame, Progress brings the frame into view unless the HIDDEN attribute for the frame or one of its ancestor frames or windows is TRUE.

- If you invoke the ENABLE statement for the parent frame of a frame family, the field representation widgets and descendant frames owned by the parent frame are all enabled. However, the field representation widgets of the descendant frames remain disabled and visually insensitive. To enable field representation widgets in the descendant frames and make them sensitive, you must invoke ENABLE statements for each of the descendant frames.

- If you specify the KEEP-TAB-ORDER option for a frame, the ENABLE statement has no affect on the tab order for the frame. Otherwise, the ENABLE statement can affect the tab order of widgets within the frame.

- The tab order for fields specified by the ENABLE statement replaces any conflicting tab order established by previous ENABLE statements or by previous settings of the FIRST-TAB-ITEM, LAST-TAB-ITEM, MOVE-AFTER-TAB-ITEM, or MOVE-BEFORE-TAB-ITEM attributes and methods.

- If you specify the ALL option with the ENABLE statement, the tab order of fields corresponds to the order they are specified in the frame definition. Also, the Data Dictionary field validations and help messages are compiled for all fields in the frame, including view-only fields (for example, text widgets).

- If you specify the ENABLE statement with *field* parameters, the specified fields are moved in the tab order to the end of the order specified for the original frame definition, and the tab order of each *field* corresponds to the order in which it is specified in the statement. The following code enables three widgets (a, b, and c) in frame A with the tab order d, e, f, a, b, and c:

```
DEFINE FRAME A a b c d e f.
ENABLE a b c WITH FRAME A.
```

Note: Note that widgets d, e, and f are not accessible until their SENSITIVE attributes are set to TRUE.

- If you use more than one ENABLE statement to enable widgets within a frame, each widget is added to the end of the tab order as it is enabled. For example, the following code enables three widgets in a frame:

```
ENABLE a.
ENABLE b.
ENABLE c.
```

  This code sets the tab order as a b c. Rearranging the ENABLE statements changes the tab order.

- For SpeedScript, these options are invalid: BGCOLOR, DCOLOR, FGCOLOR, FONT, IN-WINDOW.

See also DISABLE statement, WAIT-FOR statement

ENCODE function

Encodes a source character string and returns the encoded character string result.

Syntax

```
ENCODE ( expression )
```

expression

An expression that results in a character string value. If you use a constant, you must enclose it in quotation marks (" ").

Example

This procedure uses the ENCODE function to disguise a password that the user enters. The procedure then displays the encoded password.

r-encode.p

```
DEFINE VARIABLE password AS CHARACTER FORMAT "x(16)".
DEFINE VARIABLE id AS CHARACTER FORMAT "x(12)".
DEFINE VARIABLE n-coded-p-wrd AS CHARACTER FORMAT "x(16)".

SET id LABEL "Enter user id" password LABEL
  "Enter password" BLANK WITH CENTERED SIDE-LABELS.

n-coded-p-wrd = ENCODE(password).
DISPLAY n-coded-p-wrd LABEL "Encoded password".
```

Notes

- You can use the ENCODE function to encode a string that contains double-byte characters.
- The ENCODE function performs a one-way encoding operation that you cannot reverse. It is useful for storing scrambled copies of passwords in a database. It is impossible to determine the original password by examining the database. However, a procedure can prompt a user for a password, encode it, and compare the result with the stored, encoded password to determine if the user supplied the correct password.
- In order to ensure reliable results, the original encoding and any subsequent encoded comparisons must run in the same code page. In environments with multiple code pages, Progress strongly recommends that programs use the CODEPAGE-CONVERT function so that occurrences of the ENCODE function related to the same strings run in the same code page.
- The output of the ENCODE function is 16 characters long. Make sure the target field size is at least 16 characters long.

ENCRYPT function

Converts source data into a particular format, and returns a MEMPTR containing the encrypted data (a binary byte stream).

Note: You must use the same cryptographic algorithm, initialization vector, and encryption key values to encrypt and decrypt the same data instance.

Syntax

```
ENCRYPT ( data-to-encrypt [ , encrypt-key [ , iv-value [ , algorithm ] ] ] )
```

data-to-encrypt

The source data to encrypt. The value may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR.

encrypt-key

An optional RAW expression that evaluates to the name of the encryption key (a binary value) to use in encrypting the specified data. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-KEY attribute is used. If the value of the SYMMETRIC-ENCRYPTION-KEY attribute is also the Unknown value (?), Progress generates a runtime error.

You can generate this encryption key, based on the PKCS#5/RFC 2898 standard, by using either the GENERATE-PBE-KEY function or the GENERATE-RANDOM-KEY function.

Note: If you use the GENERATE-RANDOM-KEY function to generate an encryption key, be sure to invoke the function before invoking the ENCRYPT function (not within the ENCRYPT function, which would render the key irretrievable).

Progress compares the size of the specified encryption key to the key size specified by the cryptographic algorithm. If the key sizes are inconsistent, Progress generates a runtime error.

You are responsible for generating, storing, and transporting this value.

iv-value

An optional RAW expression that evaluates to an initialization vector value to use with the specified encryption key in the encryption operation. Using an initialization vector value increases the strength of the specified encryption key (that is, it makes the key more unpredictable). If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-IV attribute is used.

algorithm

An optional CHARACTER expression that evaluates to the name of the symmetric cryptographic algorithm to use in encrypting the specified data instance. If you specify the Unknown value (?), the current value of the SYMMETRIC-ENCRYPTION-ALGORITHM attribute is used.

For a list the supported cryptographic algorithms, see the SYMMETRIC-SUPPORT attribute reference entry.

See also DECRYPT function, SECURITY-POLICY system handle

END statement

Indicates the end of a block started with a CASE, DO, FOR EACH, PROCEDURE, or REPEAT statement or the end of an EDITING phrase or TRIGGER phrase.

Syntax

```
END
```

Example

This procedure contains two blocks, each ending with the END statement:

r-end.p

```
FOR EACH customer:
  DISPLAY customer.cust-num name phone.
  FOR EACH order OF customer:
    DISPLAY order WITH 2 COLUMNS.
  END.
END.
```

Notes

- If you do not use any END statements in a procedure, Progress assumes that all blocks end at the end of the procedure.
- If you use any END statements in a procedure, you must use one END statement for every block in the procedure.

See also

CASE statement, DO statement, EDITING phrase, FOR statement, PROCEDURE statement, REPEAT statement, Trigger phrase

ENTERED function

Checks whether a frame field has been modified during the last INSERT, PROMPT-FOR, SET, or UPDATE statement for that field, and returns a TRUE or FALSE result.

Note: Does not apply to SpeedScript programming.

Syntax

```
[ FRAME frame ] field ENTERED
```

[FRAME *frame*] *field*

The name of the frame field you are checking. If you omit the FRAME option, the field name must be unambiguous.

Example

This procedure goes through the customer table and prompts the user for a new credit-limit value. The ENTERED function tests the value the user enters. If the user enters a new value, the procedure displays the old and new credit-limit values. If the user enters the same or no value, the value does not change.

r-enter.p

```
DEFINE VARIABLE new-max LIKE credit-limit.

FOR EACH customer:
    DISPLAY cust-num name credit-limit LABEL "Current credit limit"
          WITH FRAME a 1 DOWN ROW 1.
    SET new-max LABEL "New credit limit"
          WITH SIDE-LABELS NO-BOX ROW 10 FRAME b.
    IF new-max ENTERED THEN DO:
       IF new-max <> credit-limit THEN DO:
           DISPLAY "Changing Credit Limit of" name SKIP
                   "from" credit-limit "to"
                   new-max WITH FRAME c ROW 15 NO-LABELS.
          credit-limit = new-max.
          NEXT.
       END.
    END.
    DISPLAY "No Change In Credit Limit" WITH FRAME d ROW 15.
END.
```

Notes

- If you type blanks in a field where data has never been displayed, the ENTERED function returns FALSE, a SET or ASSIGN statement does not update the underlying field or variable. Also, if Progress has marked a field as entered, and the PROMPT-FOR statement prompts for the field again and you do not enter any data, Progress no longer considers the field entered.
- If you have changed the field's window value since the last INSERT, PROMPT-FOR, SET, or UPDATE statement on that field, the ENTERED function returns FALSE. For example, if you use the DISPLAY statement to change the value of the field, ENTERED no longer returns TRUE.
- Before referencing a widget with the ENTERED function, you must scope the frame that contains that widget. For example, the following code does not compile:

```
/* This code does not compile. */

DEFINE FRAME x
   myint AS INTEGER
   mychar AS CHARACTER.

ON LEAVE OF mychar
   IF mychar ENTERED
   THEN MESSAGE "Character value changed.".

UPDATE myint mychar WITH FRAME x.
```

The DEFINE FRAME statement does not scope the frame. Therefore, the reference to the ENTERED function in the trigger cannot be evaluated. To fix the problem, reference the frame in a DISPLAY statement before the ON statement.

See also NOT ENTERED function

ENTRY function

Returns a character string entry from a list based on an integer position.

Syntax

```
ENTRY ( element , list [ , character ] )
```

element

An integer value that corresponds to the position of a character string in a list of values. If the value of *element* does not correspond to an entry in the list, Progress raises the ERROR condition. If the value of *element* is the Unknown value (?), ENTRY returns the Unknown value (?). If *element* is less than or equal to 0, or is larger than the number of elements in *list*, ENTRY returns an error.

list

A list of character strings separated with a character delimiter. The *list* can be a variable of type CHARACTER or LONGCHAR. If the value of *list* is the Unknown value (?), ENTRY returns the Unknown value (?).

character

A delimiter you define for the list. The default is a comma. This allows the ENTRY function to operate on non-comma-separated lists. If you use an alphabetic character, this delimiter is case sensitive.

Examples

This procedure returns the day of the week that corresponds to a date the user enters. The WEEKDAY function evaluates the date and returns, as an integer, the day of the week for that date. The ENTRY function uses that integer to indicate a position in a list of the days of the week.

r-entry.p

```
DEFINE VARIABLE datein AS DATE.
DEFINE VARIABLE daynum AS INTEGER.
DEFINE VARIABLE daynam AS CHARACTER INITIAL "Sunday,
        Monday, Tuesday, Wednesday, Thursday, Friday, Saturday".

SET datein LABEL "Enter a date (mm/dd/yy)".
daynum = WEEKDAY(datein).
DISPLAY ENTRY(daynum,daynam) FORMAT "x(9)" LABEL "is a" WITH SIDE-LABELS.
```

This is an example of a list separated by dashes instead of commas. The result is “helvetica”.

r-entry2.p

```
DEFINE VARIABLE typeface AS CHARACTER.typeface =
"-adobe-helvetica-bold-r-normal--*-210-*-*-*-*-iso*-*".
DISPLAY ENTRY(3, typeface, "-") FORMAT "x(16)".
```

The next procedure looks up UNIX login IDs in a small password array and returns the name of the user.

r-entry3.p

```
DEFINE VARIABLE login-name AS CHARACTER FORMAT "x(10)".
DEFINE VARIABLE real-name  AS CHARACTER FORMAT "x(20)".
DEFINE VARIABLE loop AS INTEGER.
/*username:password:uid:gid:gcos-field:home-dir:login-shell */
DEFINE VARIABLE passwd AS CHARACTER EXTENT 5 INITIAL [
  "kulig::201:120:Clyde Kulig:/users/kulig",
  "gegetskas::202:120:Neal Gegetskas:/users/geget:",
  "bertrand::203:120:Rich Bertrand:/users/bertr:",
  "lepage::204:120:Gary Lepage:/users/lepag:",
  "wnek::205:120:Jordyn Wnek:/users/wnekj:"
    ].
REPEAT:
  SET login-name.
  real-name = ?.
  DO loop = 1 TO 5:
    IF ENTRY(1,passwd[loop],":") = login-name THEN LEAVE.
  END.
  IF loop > 5 THEN
    MESSAGE "Sorry, but" login-name
      "is not in my password file.".
  ELSE
    real-name = ENTRY(5,passwd[loop],":").
  DISPLAY real-name.
END.
```

Note The ENTRY function is double-byte enabled. It can return an entry that contains double-byte characters from a specified list and the *character* delimiter can be a double-byte character.

See also LOOKUP function

ENTRY statement

Used on the left-hand side of an assignment to set the *n*th element to some value.

Syntax

```
ENTRY ( element , list [ , character ] ) = expression
```

element

An integer value that corresponds to the position of a character string in a list of values. If the value of *element* does not correspond to an entry in the list, Progress raises the ERROR condition. If the value of *element* is the Unknown value (?), ENTRY returns the Unknown value (?). If *element* is less than or equal to 0, or is larger than the number of elements in *list*, ENTRY returns an error.

list

A list of character strings separated with a character delimiter. The *list* can be a variable of type CHARACTER or LONGCHAR. If the value of *list* is the Unknown value (?), ENTRY returns the Unknown value (?).

character

A delimiter you define for the list. The default is a comma. This allows functions to operate on non-comma-separated lists. The delimiter must be only a single character. If you specify a string of more than one character, only the first character is used. If you specify a null string (""), a space character is used as the delimiter. If you use an alphabetic character, this delimiter is case sensitive.

expression

A constant, field name, variable name, or expression that results in a character string whose value you want to store in the *n*th element in a list. Progress does not pad or truncate *expression*.

Example This procedure uses three ENTRY statements:

r-ent-eq.p

```
DEFINE VARIABLE num-recs AS INTEGER.
DEFINE VARIABLE msg-txt  AS CHARACTER INITIAL
 "There are <x> records in the table.".

/* count the records. */
FOR EACH customer:
  num-recs = num-recs + 1.
END.

/* if there is only one record,
  make the message singular. */
IF num-recs = 1 THEN
  ASSIGN
    ENTRY(2,msg-txt," ") = "is"
    ENTRY(4,msg-txt," ") = "record".

/* insert the record count into the string. */
ENTRY(3,msg-txt," ") = STRING(num-recs).

MESSAGE msg-txt.
```

Note The ENTRY statement is double-byte enabled. It can insert an entry that contains double-byte characters into a specified list and the *character* delimiter can be a double-byte character.

See also ENTRY function

EQ or = operator

Returns a TRUE value if two expressions are equal.

Syntax

```
expression { EQ | = } expression
```

expression

A constant, field name, variable name, or expression. The expressions on either side of the EQ or = must be of the same data type, although one might be an integer and the other a decimal.

Example

This procedure prompts for the initials of a sales rep. The FOR EACH block reads all the order records for that sales rep. The DISPLAY statement displays information from each of the retrieved records.

r-eq.p

```
PROMPT-FOR order.sales-rep WITH SIDE-LABELS CENTERED.

FOR EACH order WHERE sales-rep EQ INPUT sales-rep:
  DISPLAY order-num cust-num order-date promise-date ship-date
    WITH CENTERED.
END.
```

Notes

- By default, Progress uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).

- You can compare character strings with EQ. Most character comparisons are case insensitive in Progress. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *expression* is a field or variable defined as case sensitive, the comparison is case sensitive and “Smith” does not equal “smith”.

- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ’ , but lowercase a is greater than these.

- If one of the expressions has an Unknown value (?) and the other does not, the result is FALSE. If both have the Unknown value (?), the result is TRUE. However, for SQL, if the value of either or both expressions is the Unknown value (?), then the result is the Unknown value (?).

- The equal comparison ignores trailing blanks. Thus, “abc” is equal to “abc ”. However, leading and embedded blanks are treated as characters and “ abc” is not equal to “abc”.

- You can use EQ to compare DATE, DATETIME, and DATETIME-TZ data. The data type that contains less information (that is, a DATE value contains less information than a DATETIME value, and a DATETIME value contains less information than a DATETIME-TZ value) is converted to the data type with more information by setting the time value to midnight, and the time zone value to the session's time zone (when the data type does not contain the time or time zone). Comparisons with DATETIME-TZ data are based on Coordinated Universal Time (UTC) date and time.

- You can use EQ to compare one BLOB field to another. Progress performs a byte-by-byte comparison.

- You can use EQ to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or Progress raises a run-time error.

- You can use EQ to compare a CLOB field only to the Unknown value (?).

ERROR function

Indicates whether an error occurred during a FILL or SAVE-ROW-CHANGES operation on the specified ProDataSet temp-table buffer.

Syntax

```
ERROR( buffer-name )
```

buffer-name

The name of a ProDataSet temp-table buffer.

Notes

- The ERROR function corresponds to the ERROR attribute.
- You can invoke the ERROR function from within a WHERE clause (unlike the corresponding attribute).

ETIME function

Returns the time (in milliseconds) elapsed since the OpenEdge session began or since ETIME (elapsed time) was last set to 0. To set ETIME to 0, pass it a positive logical value, such as YES or TRUE.

Syntax

```
ETIME [ ( logical ) ]
```

logical

A logical value, such as YES or TRUE. The default value is NO.

Examples

This procedure displays the time that elapsed since you began your OpenEdge session:

r-etime.p

```
DISPLAY ETIME.
```

This procedure sets ETIME to 0, runs a procedure called `applhelp.p`, and displays the elapsed time, which, in this case, equals the time required to execute `applhelp.p`:

r-etime2.p

```
DEFINE VARIABLE a AS INTEGER.

DO:
  a = ETIME(yes).
  RUN applhelp.p.
  DISPLAY ETIME.
END.
```

Notes

- ETIME is accurate to at least one-sixtieth of a second, but accuracy varies among systems.
- Progress resets ETIME during startup, not immediately after you enter the pro command. Therefore, the time returned is only an approximation of the time elapsed since your session began.

See also

TIME function

EXP function

Returns the result of raising a number to a power. The number is called the *base* and the power is called the *exponent*.

Syntax

```
EXP ( base , exponent )
```

base

A constant, field name, variable name, or expression that evaluates to a numeric value.

exponent

A numeric expression.

Example

This procedure calculates how much a principal amount invested at a given compounded annual interest rate grows over a specified number of years:

r-exp.p

```
DEFINE VARIABLE principal AS DECIMAL
  FORMAT "->>>,>>9.99" LABEL "Amt Invested".
DEFINE VARIABLE rate AS INTEGER FORMAT "->9"
  LABEL "Interest %".
DEFINE VARIABLE num-yrs AS INTEGER FORMAT ">>9"
  LABEL "Number of Years".
DEFINE VARIABLE final-amt AS DECIMAL FORMAT
  "->>>,>>>,>>>,>>>,>>>,>>9.99" LABEL "Final Amount".
REPEAT:
  UPDATE principal rate num-yrs.
  final-amt = principal * EXP(1 + rate / 100,num-yrs).
  DISPLAY final-amt.
END.
```

Notes

- After converting the base and exponent to the floating-point format, the EXP function uses standard system library routines. On some machines, these routines do not handle large numbers well and might cause your terminal to hang. Also, because the calculations are done in floating-point arithmetic, full decimal precision is not possible beyond 1-12 significant digits on most machines.
- The EXP function is precise to approximately 10 decimal points.

EXPORT statement

Converts data to a standard character format and displays it to the current output destination (except when the current output destination is the screen) or to a named output stream. You can use data exported to a file in standard format as input to other Progress procedures.

Syntax

```
EXPORT [ STREAM stream ] [ DELIMITER character ]
  {    expression ...
     | record [ EXCEPT field ... ]
  }
  [ NO-LOBS ]
```

```
EXPORT [ STREAM stream ] { memptr | longchar }
```

STREAM *stream*

The name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

DELIMITER *character*

The character to use as a delimiter between field values. The *character* parameter must be a quoted **single** character. The default is a space character.

If you specify more than one character as a delimiter, Progress uses the first character as the delimiter.

expression . . .

One or more expressions that you want to convert into standard character format for display to an output destination.

record

The name of the record buffer with fields that you want to convert into the standard character format to display to an output destination.

To use EXPORT with a record in a table name used in multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

EXCEPT *field* **. . .**

Progress exports all fields except those fields listed in the EXCEPT phrase.

memptr

A variable of data type MEMPTR that contains the text to export. The EXPORT statement may contain a MEMPTR in its field list as long as it is the only field in the list.

longchar

A variable of data type LONGCHAR that contains the text to export. The EXPORT statement may contain a LONGCHAR in its field list as long as it is the only field in the list.

NO-LOBS

Directs Progress to ignore large object data when exporting records that contain BLOB or CLOB fields.

Examples

This procedure converts the data in the customer table into standard character format and sends that data to the customer.d file:

r-exprt.p

```
OUTPUT TO customer.d.
FOR EACH customer:
  EXPORT customer.
END.

OUTPUT CLOSE.
```

The next procedure shows how each EXPORT statement creates one line of data. That is, fields are not wrapped onto several lines.

r-exprt2.p

```
OUTPUT TO custdump.
FOR EACH customer:
  EXPORT cust-num name credit-limit.
END.

OUTPUT CLOSE.
```

That procedure creates a text file, `custdump`, with one line for each customer. This is a typical line of output:

```
1 "Lift Line Skiing" 58400
```

Use the DELIMITER option to specify a character other than a space to separate fields in the output file. For example, the following procedure uses a semicolon:

r-cstout.p

```
OUTPUT TO custdump2.

FOR EACH customer:
  EXPORT DELIMITER ";" cust-num name credit-limit.
END.

OUTPUT CLOSE.
```

This is a typical line of output from this code:

```
1;"Lift Line Skiing";58400
```

The following example displays using a MEMPTR to EXPORT mixed character and binary data:

r-expmem.p

```
/* character and binary data mixed */

DEFINE VARIABLE z AS MEMPTR.

SET-SIZE(z) = 100.

PUT-STRING(z,1) = "hi there".
PUT-LONG(z,10) = 235.
PUT-STRING(z,14) = "afterint".
PUT-LONG(z,22) = 76.

OUTPUT TO abc BINARY NO-CONVERT.
EXPORT z.
OUTPUT CLOSE.
```

Notes

- The EXPORT statement must follow an OUTPUT TO statement, which redirects the output destination.
- Other procedures can use the data exported with the EXPORT statement as input by reading the file with the INSERT, PROMPT-FOR, SET, UPDATE or IMPORT statements, naming one field or variable to correspond to each data element.
- The data is in a **standard format** to be read back into Progress. All character fields are enclosed in quotes ("") and quotes contained in the data you are exporting are replaced by two quotes (""). A single space separates one field from the next. An Unknown value (?) is displayed as an unquoted question mark (?).
- There are no trailing blanks, leading zeros, or formatting characters (for example, dollar signs) in the data.
- Progress exports logical fields as the value YES or NO.
- A Format phrase with an EXPORT statement is ignored.
- If you use a single qualified identifier with the EXPORT statement, the Compiler first interprets the reference as *dbname.tablename*. If the Compiler cannot resolve the reference as *dbname.tablename*, it tries to resolve it as *tablename.fieldname*.

- When exporting fields, you must use table names that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.
- When exporting RECID fields, you must explicitly state the RECID field name in the EXPORT statement.
- When exporting ROWID variables or fields in a work table, you must convert the ROWID variable or field to a character string using the STRING function.
- When exporting records that contain a BLOB or CLOB field, Progress creates a separate object data file using a unique filename with a `.blb` extension and stores that filename in the BLOB or CLOB field of the exported record. (When importing records that contain a BLOB or CLOB field, Progress uses this filename to locate the object data file associated with each record.) If the BLOB or CLOB field contains the Unknown value (`?`), Progress stores the Unknown value (`?`) in the BLOB or CLOB field of the exported record, and does not create an object data file. If the BLOB or CLOB field contains a zero-length object, Progress creates a zero-length object data file.

  Progress raises the ERROR condition if an object data file cannot be created.
- The EXPORT statement creates large object data files in the directory specified as the output destination in the OUTPUT TO statement, by default. You can use the `LOB-DIR` option on the OUTPUT TO statement to specify the directory in which the EXPORT statement creates the BLOB and CLOB data files.
- Use the `NO-LOBS` option with the EXPORT statement to ignore large object data when exporting records that contain BLOB or CLOB fields. When you specify the `NO-LOBS` option, Progress stores the Unknown value (`?`) in the BLOB or CLOB field of the exported records and does not create the associated object data files.
- When exporting DATETIME and DATETIME-TZ data, the data format is fixed and conforms to the ISO 8601 standard for date/time representations (`YYYY-MM-DDTHH:MM:SS.SSS+HH:MM`). For DATETIME, there is no time zone offset.
- If you use the DELIMITER option of the EXPORT statement to specify a delimiter other than a space character, you must specify the same delimiter character in a subsequent IMPORT statement that loads the data.

- EXPORT is sensitive to the Date format (-d), Century (-yy), and European numeric (-E) startup parameters. When loading data with the IMPORT statement, use the same settings that you used with the EXPORT statement.
- In the MEMPTR version of the EXPORT statement, the MEMPTR's size will determine how much is written to the file. If the size of a MEMPTR is 100, and it only contains a string of length 10, the entire 100 bytes will still be written to the file. The PUT-BYTES statement and GET-BYTES function may be used to move portions of MEMPTRs to areas with varying sizes.You can read and write parts of a file by using MEMPTRs of varying sizes, and multiple EXPORT/IMPORT statements on the same file.
- When exporting a LONGCHAR variable, Progress also stores the variable's code page in the file header.

See also DEFINE STREAM statement, DISPLAY statement, IMPORT statement, OUTPUT CLOSE statement, OUTPUT TO statement, PUT statement, STRING function

EXTENT function

This function returns the extent of an array field or variable. More specifically, it returns:

- The constant or variable extent value for a field or variable defined as a determinate array.
- The Unknown value (?) for a field or variable defined as an unfixed indeterminate array.
- The extent for a field or variable defined as a fixed indeterminate array.
- Zero for a field or variable that is not an array.

Note: The EXTENT function corresponds to the EXTENT attribute.

Syntax

```
EXTENT ( array )
```

array

Any array field or variable.

Example

In the following example, the EXTENT function is used to set the limit of a DO loop that cycles through all elements of an array:

r-arrext.p

```
DEFINE VARIABLE int_value AS INTEGER EXTENT 3 INITIAL [1, 2, 3].

DEFINE VARIABLE i     AS INTEGER.
DEFINE VARIABLE tot   AS INTEGER LABEL "The total is".

DO i = 1 TO EXTENT(int_value):
   tot = tot + int_value[i].
END.

DISPLAY tot.
```

See also

DEFINE VARIABLE statement, ENTRY function, EXTENT attribute

FILL function

Generates a character string made up of a character string that is repeated a specified number of times.

Syntax

```
FILL ( expression , repeats )
```

expression

An expression that yields a character value. This expression can contain double-byte characters.

repeats

A constant, field name, variable name, or expression with a value of integer. The FILL function uses this value to repeat the *expression* you specify. If the value of *repeats* is less than or equal to 0, FILL produces a null string.

Example

This example procedure produces a bar chart that depicts each customer's balance as a percentage of the total of all outstanding balances. The first FOR EACH block accumulates the value of balance for each customer, producing a total balance value for all customers. The next FOR EACH block goes through the customer table again, figuring each customer's balance as a percentage of the total.

r-fill.p

```
/* DEFINE VARIABLE percentg AS INTEGER FORMAT ">>9".
DEFINE VARIABLE fillchar AS CHARACTER FORMAT "x".

fillchar =  "*".

FOR EACH customer:
  ACCUMULATE balance (TOTAL).
END.

DISPLAY "Percentage of Outstanding Balance" WITH CENTERED NO-BOX.

FOR EACH customer WHERE balance > 0:
  percentg = balance / (ACCUM TOTAL balance) * 100.
  FORM SKIP  name percentg LABEL "%" bar AS CHAR
    LABEL "  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17"
    FORMAT "x(50)" WITH NO-BOX NO-UNDERLINE USE-TEXT.
  COLOR DISPLAY BRIGHT-RED bar.
  DISPLAY name percentg FILL(fillchar,percentg * 3) @ bar.
END.
```

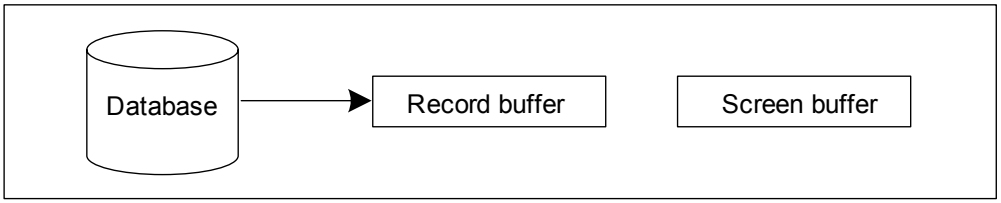
The FORM statement describes the frame layout, including the name, the percentage of total balance, and a bar across the top of the frame. (The bar variable is defined on-the-fly; it has no corresponding DEFINE VARIABLE statement at the top of the procedure. It is defined in the FORM statement and has its own label and format.) The DISPLAY statement following the FORM statement displays the bar variable. If the procedure is running on UNIX or on a monochrome PC monitor, Progress ignores the COLOR BRIGHT-RED. However, if the procedure is running on a PC with a color monitor, the bar is displayed in BRIGHT-RED (a predefined color on the PC). The final DISPLAY statement displays the bars.

The `fillchar` assignment statement sets the fill character to asterisk (*). The FILL function generates a string made up of fill characters that is the percentage of total sales multiplied by three (each percentage point uses three fill characters).

FIND statement

Locates a single record in a table and moves that record into a record buffer.

Data movement



Syntax

```
FIND [ FIRST | LAST | NEXT | PREV ] record
     [ constant ]
     [ OF table ]
     [ WHERE expression ]
     [ USE-INDEX index ]
     [ USING [ FRAME frame ] field
        [ AND [ FRAME frame ] field ] ...
     ]
     [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
     [ NO-WAIT ]
     [ NO-PREFETCH ]
     [ NO-ERROR ]
```

You can specify the OF, WHERE, USE-INDEX, and USING options in any order.

```
FIND CURRENT record
     [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
     [ NO-WAIT ]
     [ NO-ERROR ]
```

FIRST

Finds the first record in the table that meets the characteristics you might have specified with *record*. If the buffer named in the *record* was preselected in a DO or REPEAT statement, FIND locates the first record in that preselected subset of records.

LAST

Finds the last record in the table that meets the specified characteristics of the *record*. If the buffer named in the *record* was preselected in a DO or REPEAT statement, FIND locates the last record in that preselected subset of records.

NEXT

Finds the next record in the table that meets the specified characteristics of the *record*. If no record has been found, FIND NEXT behaves like FIND FIRST. If the buffer named in the *record* was preselected in a DO or REPEAT statement, FIND locates the next record in that preselected subset of records.

PREV

Finds the previous record in the table. If no record has yet been found, FIND PREV behaves like FIND LAST. If the buffer named in the *record* was preselected in a DO or REPEAT statement, FIND locates the previous record in that preselected subset of records.

CURRENT

Refetches the current record in the buffer with the specified lock status.

record

Identifies the record you want to retrieve. The *record* parameter can be a reference to a database table or a defined buffer.

constant

The value of a single component, unique, primary index for the record you want.

```
FIND customer 1.
```

Progress converts this FIND statement with the *constant* option of 1.

```
FIND customer WHERE cust-num = 1.
```

The cust-num field is the only component of the primary index of the customer table. If you use the *constant* option, you must use it once in a single Record phrase, and it must precede any other options in the Record phrase.

OF *table*

Qualifies the records by relating the record to a record in another table.

```
PROMPT-FOR order.order-num.
FIND order USING order-num.
DISPLAY order.
FIND customer OF order.
DISPLAY customer.
```

The OF option relates the order table to the customer table, telling Progress to select the customer record related to the order record currently being used. When you use OF, all fields participate in match criteria, if an index is multi-field. The relationship is based on having a UNIQUE index in one table. Progress converts the FIND statement with the OF option to the following:

```
FIND customer WHERE customer.cust-num = order.cust-num.
```

You can access related tables using WHERE, whether or not the field names of the field or fields that relate the tables have the same name.

WHERE *expression*

Qualifies the records you want to access. The *expression* is a constant, field name, variable name, or expression whose value you want to use to select records. You can use the WHERE keyword even if you do not supply an *expression.*

```
FOR EACH customer WHERE {*}
```

The WHERE clause may not work the same way against a DataServer as it does against the OpenEdge database. Refer to the appropriate DataServer Guide, *OpenEdge Data Management: DataServer for ODBC* or *OpenEdge Data Management: DataServer for ORACLE*, for additional information on how this feature will perform.

Note: You cannot reference a BLOB or CLOB field in a WHERE clause.

USE-INDEX *index*

Identifies the index you want to use while selecting records. If you do not use this option, Progress selects an index to use based on the criteria specified with the WHERE, USING, OF, or *constant* options.

USING [FRAME *frame*] *field* [AND [FRAME *frame*] *field*] . . .

One or more names of fields for selecting records. The field you name in this option must have been entered previously, usually with a PROMPT-FOR statement. The field must be viewed as a fill-in or text widget.

The USING option translates into an equivalent WHERE option.

```
PROMPT-FOR customer.cust-num.
FIND customer USING cust-num.
```

This FIND statement is the same as the following statement:

```
FIND customer WHERE customer.cust-num = INPUT customer.cust-num.
```

The cust-num field is a non-abbreviated index. However, look at this example:

```
PROMPT-FOR customer.name.
FIND customer USING cust-name.
```

If the name field is an abbreviated index of the customer table, Progress converts the FIND statement with the USING option into this following statement:

```
FIND customer WHERE customer.name BEGINS INPUT name.
```

Note that field can be expanded to be FRAME *frame field.*

SHARE-LOCK

Tells Progress to put a SHARE-LOCK on records as they are read. Other users can still read a record that is share locked, but they cannot update it. By default, Progress puts a SHARE-LOCK on a record when it is read, and automatically puts an EXCLUSIVE-LOCK on a record when it is modified (unless the record is already EXCLUSIVE-LOCKed).

If you use the SHARE-LOCK option and Progress tries to read a record that is EXCLUSIVE-LOCKed by another user, Progress waits to read the record until the EXCLUSIVE-LOCK is released. Progress displays a message to the user of that procedure, identifying the table that is in use, the user ID of the user, and the tty of the terminal using the table.

If you are using a record from a work table, Progress disregards the SHARE-LOCK option.

EXCLUSIVE-LOCK

Tells Progress to put an EXCLUSIVE-LOCK on records as they are read. Other users cannot read or update a record that is EXCLUSIVE-LOCKed, except by using the NO-LOCK option. They can access that record only when the EXCLUSIVE-LOCK is released. Progress automatically puts a SHARE-LOCK on a record when it is read and automatically puts an EXCLUSIVE-LOCK on a record when it is updated.

If a record is read specifying EXCLUSIVE-LOCK, or if a lock is automatically changed to EXCLUSIVE-LOCK by an update, a user's read or update will wait if any other user has the record SHARE-LOCKed or EXCLUSIVE-LOCKed.

When a procedure tries to use a record that is EXCLUSIVE-LOCKed by another user, Progress displays a message identifying the table that is in use, the user ID of the user, and the tty of the terminal using the table.

If you are using a record from a work table, Progress disregards the EXCLUSIVE-LOCK option.

NO-LOCK

Tells Progress to put no locks on records as they are read, and to read a record even if another user has it EXCLUSIVE-LOCKed.

Other users can read and update a record that is not locked. By default, Progress puts a SHARE-LOCK on a record when it is read (unless it is using a CAN-FIND function, which defaults to NO-LOCK), and automatically puts an EXCLUSIVE-LOCK on a record when it is updated (unless the record is already EXCLUSIVE-LOCKed). A record that has been read NO-LOCK must be reread before it can be updated, as shown in this example:

```
DEFINE VARIABLE rid AS ROWID.
FIND FIRST customer NO-LOCK.
rid = ROWID(customer).
FIND customer WHERE
  ROWID(customer) = rid EXCLUSIVE-LOCK.
```

If a procedure finds a record and it places it in a buffer using NO-LOCK and you then re-fund that record using NO-LOCK, Progress does not reread the record. Instead, it uses the copy of the record that is already stored in the buffer.

When you read records with NO-LOCK, you have no guarantee of the overall consistency of those records because another user might be in the process of changing them. For example, when a record is updated, changes to indexed fields are written immediately, but changes to other fields are deferred. In the meantime, the record is in an inconsistent state. For example, the following procedure might display a cust-num of 0 if another user's active transaction has created a record and assigned a value to the indexed field cust-num that is greater than 100.

```
FOR EACH customer WHERE cust-num > 100 NO-LOCK:
  DISPLAY cust-num.
END.
```

If you are using a record from a work table, Progress disregards the NO-LOCK option.

NO-WAIT

Causes FIND to return immediately and raise an error condition if the record is locked by another user (unless you use the NO-ERROR option on the same FIND statement). For example:

```
FIND customer USING cust-name NO-ERROR NO-WAIT.
```

Without the NO-WAIT option, Progress waits until the record is available.

Progress ignores NO-WAIT when it is used with work tables and databases that are only accessed by a single user.

NO-PREFETCH

Specifies that only one record is sent across the network at a time. If you are accessing a remote server and do not specify this option, Progress might send more than one record from the server to the client in each network packet. Sending more than one packet may, in rare cases, create inconsistencies with Progress Version 6 or earlier.

NO-ERROR

Tells Progress not to display error messages for any errors it might encounter. Instead, error information is passed the ERROR-STATUS system handle. Possible errors include: not finding a record that satisfies the *record-phrase,* finding more than one record that satisfies the *record-phrase* for a unique find, or finding a record that is locked with the NO-WAIT option on the FIND. If you use the NO-ERROR option, you can also use the AVAILABLE function to test if FIND found a record.

Examples

This procedure produces a report that shows all the customers who bought a particular item, and the quantity that they bought. The procedure finds an item record, the order-lines that use that item, the order associated with each order-line, and the customer associated with each order.

r-find.p

```
REPEAT:
     PROMPT-FOR item.item-num.
      FIND item USING item-num.
      DISPLAY item-num item-name.
     REPEAT:
         FIND NEXT order-line OF item.
         FIND order OF order-line.
         FIND customer WHERE customer.cust-num = order.cust-num.
         DISPLAY customer.name order.order-num order-line.qty (TOTAL).
      END.
END.
```

The FIND FIRST statement in the following procedure finds the first record with a name field value that alphabetically follows the name supplied by the user. The FIND NEXT statement uses the name index to find the next record in the table, using the name index.

r-find2.p

```
DEFINE VARIABLE start-name LIKE customer.name.
REPEAT:
  SET start-name.
  FIND FIRST customer WHERE name >= start-name.
  REPEAT:
    DISPLAY name.
    FIND NEXT customer USE-INDEX name.
  END.
END.
```

Notes

- If a FIND statement fails, it indicates that the buffer named in *record* contains no record.
- If Progress finds an old record in the record buffer when executing a FIND, it validates the record then writes it out. (If the record fails validation, Progress returns an error message.) Then it clears the buffer and stores the located record in the record buffer.
- A FIND statement that does not supply FIRST, LAST, NEXT, or PREV is a unique FIND and must be able to locate, at most, one record based solely on the conditions in the expression or WHERE clause it is using.
- Fields referenced in the WHERE clause do not have to be indexed.

- WHERE conditions can include Boolean operations.
- If a FIND NEXT or FIND PREV does not find another record, Progress takes the end-key action. By default, this action is UNDO, LEAVE for a FOR EACH, REPEAT, or procedure block.
- See the DEFINE BUFFER statement reference entry for a description of how to use FIND on a PRESELECTed set of records.
- When you use the FIND statement, Progress selects an index to use based on the WHERE condition or the USE-INDEX option.
- Your position in an index is established when you find a record and is only modified by subsequent record retrievals, not by CREATEs or by changing indexed field values.
- If you are using the FIND statement to find a record in a work table, you must use the FIRST, LAST, NEXT, or PREV option with the FIND statement.
- In a REPEAT block, if you use the FIND NEXT statement to find a record and then do an UNDO, RETRY of a block, the FIND NEXT statement reads the next record in the table, rather than the one found in the block iteration where the error occurred:

```
REPEAT:
  FIND NEXT order.
  DISPLAY order.
  SET order-num.
  SET order-date promise-date.
END.
```

Progress does an UNDO, RETRY if there is an error and you explicitly use the UNDO, RETRY statement, or if you press **END-ERROR** on the second or later windows interaction in a block.

Here, if you press **END-ERROR** during the second SET statement, Progress displays the next record in the table.

If you are using a FOR EACH block to read records, and do an UNDO, RETRY during the block, you see the same record again rather than the next record.

If you want to use a REPEAT block and want to see the same record in the event of an error, use the RETRY function:

```
REPEAT:
  IF NOT RETRY THEN FIND NEXT order.
  DISPLAY order.
  SET order-num.
  SET order-date promise-date.
END.
```

- When you use FIND NEXT or FIND PREV to find a record after updating another record, be careful not to lose your updates in case the record you want to find is unavailable.

```
FIND FIRST customer.
REPEAT:
  UPDATE customer.
  FIND NEXT customer.
END.
```

In this example, if the FIND NEXT statement fails to find the customer record, any changes made during the UPDATE statement are undone. To avoid this, use the following technique:

```
FIND FIRST customer.
REPEAT:
  UPDATE customer.
  FIND NEXT customer NO-ERROR.
  IF NOT AVAILABLE customer THEN LEAVE.
END.
```

- After you use the FIND LAST statement to find the last record in a table, Progress positions the index cursor on that record. Any references within the same record scope to the next record fail.

```
FIND LAST customer.
RELEASE customer.
DISPLAY AVAILABLE customer.
REPEAT:
  FIND NEXT customer.
  DISPLAY name.
END.
```

In this example, the RELEASE statement releases the last customer record from the customer record buffer and the following DISPLAY statement displays FALSE because the customer record is no longer available. However, the index cursor is still positioned on that last record. Therefore, the FIND NEXT statement fails.

- If you use FIND . . . WHERE ROWID *rowid = . . .* on a PRESELECTed list of records, the temporary preselect index cursor is not reset. So, FIND NEXT does not find the record that follows record *rowid* in the preselected list. (See the DO statement and REPEAT statement reference entries for details.)

- When you use a FIND NEXT or FIND PREV statement in a subprocedure to access a record from a shared buffer, keep the following in mind:

  – When you run a Progress procedure, Progress creates a **cursor indicator** for each index accessed through a FIND statement in the procedure and each NEW buffer defined in the procedure. A cursor indicator serves as an anchor for index cursors associated with a table or buffer. An index cursor is attached to the cursor indicator when you enter a block of code where a record buffer is scoped. If two different indexes are used for the same record buffer within a single block of code, two index cursors are attached to the same cursor indicator. When the program control leaves the block where a record buffer is scoped, all index cursors attached to the cursor indicator are released.

  – When Progress encounters a subprocedure in a procedure, it constant, field name, variable name, or checks through the existing index cursors before creating any other index cursors required by the statements in the subprocedure.

  – If the USE-INDEX of the FIND NEXT or FIND PREV statement in a subprocedure accesses an index cursor for a shared buffer that existed prior to the beginning of the subprocedure, the FIND NEXT or FIND PREV statement returns the next or previous record for the shared buffer, based upon the last record found in that buffer and the USE-INDEX of the FIND statement.

  – If the USE-INDEX of the FIND NEXT or FIND PREV statement in a subprocedure accesses an index cursor created for a shared buffer at the beginning of the subprocedure, the FIND NEXT or FIND PREV statement returns the first or last record for the shared buffer, based upon the USE-INDEX of the FIND statement.

- If a field or variable referenced with FIND is used in more than one frame, then Progress uses the value in the frame most recently introduced in the procedure. To make sure you are using the appropriate frame, use the FRAME option with the FIND function to reference a particular frame.

- When a FIND statement executes, any FIND trigger defined for the table is executed.
- The FIND CURRENT statement is useful for maintaining small transaction size in updates. For an example, see the CURRENT-CHANGED function reference entry.
- FIND triggers do not execute for a FIND CURRENT statement.
- Progress does not allow a FIND statement within a FOR EACH block unless you specify a different table than the one referenced in the FOR EACH block. When you attempt to compile the following example, Progress returns the error message "FIND cannot be processed for a FOR EACH mode record."

```
FOR EACH customer:
 FIND CURRENT customer.
END.
```

- Progress restricts the FIND statement within a PRESELECT block in the following situations:
  - You cannot specify a lock option on the FIND statement. You must specify it in the PRESELECT phrase. Attempting to compile the following example produces the error message "LOCK keyword illegal on FIND within a PRESELECT for the same table".

```
DO PRESELECT EACH customer:
 FIND NEXT customer NO-LOCK.
END.
```

  - You cannot specify a unique FIND or a FIND CURRENT for the same table. The following example produces the error message "Unique FIND not allowed within a PRESELECT on the same table" when you try to compile it.

```
DO PRESELECT EACH customer:
 FIND customer 5.
END.
```

See also AMBIGUOUS function, AVAILABLE function, CAN-FIND function, CURRENT-CHANGED function, DEFINE BUFFER statement, ERROR-STATUS system handle, FOR statement,GET statement, LOCKED function, NEW function, PRESELECT phrase

FIRST function

Returns a TRUE value if the current iteration of a DO, FOR EACH, or REPEAT . . . BREAK block is the first iteration of that block.

Syntax

```
FIRST ( break-group )
```

break-group

The name of a field or expression you name in the block header with the BREAK BY option.

Example

The `r-first.p` procedure displays the order number, order-lines on the order, the extended price of each order-line, and a total order value for each order record:

r-first.p

```
DEFINE VARIABLE order-value AS DECIMAL LABEL "Order-value".

FOR EACH order:
  DISPLAY order-num.
  FOR EACH order-line OF order BREAK BY qty * price:
    IF FIRST(qty * price) THEN order-value = 0.
    order-value = order-value + qty * price.
    DISPLAY line-num item-num qty * price
      LABEL "Extended-price".
  END.
  DISPLAY order-value.
END.
```

Because the inner FOR EACH block iterates until Progress reads all the order-lines, the procedure must set the order-value variable to 0 each time a new order is used in that block. The FIRST function uses the (`qty * price`) expression as the *break-group* to keep track of whether or not the current iteration is the first iteration of the FOR EACH block.

See also

DO statement, FIRST-OF function, FOR statement, LAST function, LAST-OF function, REPEAT statement

FIRST-OF function

Returns a TRUE value if the current iteration of a DO, FOR EACH, or REPEAT . . . BREAK block is the first iteration for a new break group, and modifies all three block types.

Syntax

```
FIRST-OF ( break-group )
```

break-group

The name of a field or expression you name in the block header with the BREAK BY option.

Example

This procedure generates a report that lists all the item records grouped by catalog page. When the cat-page value changes, the procedure clears the current list of items and displays items belonging to the new catalog page. The FIRST-OF function uses the value of the cat-page field to determine when that value is different from the value during the last iteration.

r-firstf.p

```
FOR EACH item BREAK BY Cat-Page:
      IF FIRST-OF(Cat-Page) THEN CLEAR ALL.
      DISPLAY Cat-Page item-num item-name.
END.
```

Note

When you calculate in a block use the BREAK option to tell Progress to calculate when the value of certain expressions changes. Progress uses default formatting to display the results of these calculations. To control the formatting, use the FIRST-OF function to determine the start of a break group and then change the formatting.

See also

DO statement, FIRST function, FOR statement, LAST function, LAST-OF function, REPEAT statement

FIX-CODEPAGE function

Sets the code page of an empty LONGCHAR variable. When set to a valid code page, the code page of the specified variable is fixed and overrides any default behavior in assignment operations (including the COPY-LOB, OVERLAY, and SUBSTRING statements).

Syntax

```
FIX-CODEPAGE ( longchar ) = codepage
```

longchar

The name of a LONGCHAR variable. The variable must be set to the Unknown value (?) or the empty string (""). If the string length is greater than 0, Progress returns an error.

codepage

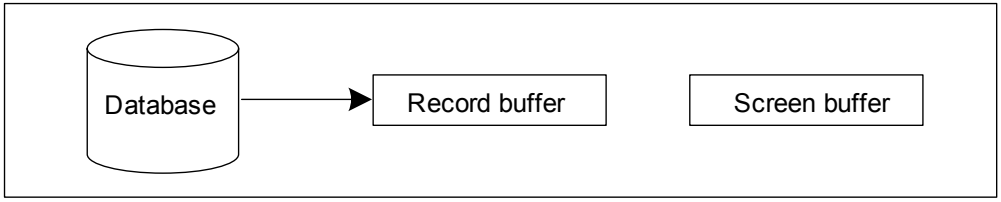
A character expression that evaluates to the name of a code page. The name you specify must be a valid code page name available in `DLC/convmap.cp`. If *codepage* is the Unknown value (?), the code page of the LONGCHAR variable is not fixed.

See also GET-CODEPAGE function, IS-CODEPAGE-FIXED( ) function

FOR statement

Starts an iterating block that reads a record from each of one or more tables at the start of each block iteration.

Data movement



Block properties

Iteration, record reading, record scoping, frame scoping, transactions by default.

Syntax

```
[ label: ]
FOR [ EACH | FIRST | LAST ] record-phrase
[ , [ EACH | FIRST | LAST ] record-phrase ] ...
[ query-tuning-phrase ]
[ BREAK ]
[ BY expression [ DESCENDING ]
| COLLATE ( string , strength [ , collation ] ) [ DESCENDING ]
] ...
[ variable = expression1 TO expression2 [ BY k ] ]
[ WHILE expression ]
[ TRANSACTION ]
[ on-error-phrase ]
[ on-endkey-phrase ]
[ on-quit-phrase ]
[ on-stop-phrase ]
[ frame-phrase ]
```

EACH

Starts an iterating block, finding a single record on each iteration. If you do not use the EACH keyword, the Record phrase you use must identify exactly one record in the table.

FIRST

Uses the criteria in the *record-phrase* to find the first record in the table that meets that criteria. Progress finds the first record before any sorting.

```
FOR FIRST customer BY credit-limit:
  DISPLAY customer.
END.
```

The previous procedure displays customer 1 (cust-num is the primary index of the customer table), not the customer with the lowest credit-limit. A procedure that displays the customer with the lowest credit-limit looks like the following:

```
FOR EACH customer BY credit-limit:
  DISPLAY customer.
  LEAVE.
END.
```

See the Notes section for more information on using this option.

LAST

Uses the criteria in the *record-phrase* to find the last record in the table that meets that criteria. Progress finds the last record before sorting.

```
FOR LAST customer BY credit-limit:
  DISPLAY customer.
END.
```

The procedure above displays the customer with the highest customer number (cust-num is the primary index of the customer table), not the customer with the highest credit-limit.

A procedure that displays the customer with the highest credit-limit looks like the following:

```
FOR EACH customer BY credit-limit DESCENDING:
  DISPLAY customer.
  LEAVE.
END.
```

See the Notes section for more information on using this option.

record-phrase

Identifies the set of records you want to retrieve. This can also be the built-in buffer name, `proc-text-buffer`, that you can use to return table rows from a stored procedure.

To use FOR EACH/FIRST/LAST to access a record in a table defined for multiple databases, you must qualify the record's table name with the database name.

This is the syntax for *record-phrase*:

Syntax

```
record
  [ constant ] [ OF table ]
  [ USE-INDEX index ]
  [ USING [ FRAME frame ] field
      [ AND [ FRAME frame ] field ] ...
  ]
  [ WHERE expression ]
  [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
  [ NO-PREFETCH ]
```

Specifying multiple occurrences of *record-phrase* selects the tables using an inner join.

For more information on *record-phrase* and inner joins, see the Record phrase reference entry.

query-tuning-phrase

Allows programmatic control over the execution of a DataServer query. Following is the syntax for the *query-tuning-phrase*:

Syntax

```
QUERY-TUNING
  (
    { [    LOOKAHEAD [ CACHE-SIZE integer ]
         | NO-LOOKAHEAD
      ]
      [ DEBUG { SQL | EXTENDED } | NO-DEBUG ]
      [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
      [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
      [ BIND-WHERE | NO-BIND-WHERE ]
      [ INDEX-HINT | NO-INDEX-HINT ]
    }
  )
```

For more information on the *query-tuning-phrase*, see *OpenEdge Data Management: DataServer for ODBC*, *OpenEdge Data Management: DataServer for ORACLE*, and *OpenEdge Data Management: DataServer for Microsoft SQL Server*.

BREAK

Over a series of block iterations, you might want to do some work based on whether the value of a certain field changes. This field defines a break group. For example, you might be accumulating some value, such as a total. You use the BREAK option to define state as the break group. For example:

```
FOR EACH customer BREAK BY state:
  DISPLAY state name credit-limit (TOTAL BY state).
END.
```

Here, Progress accumulates the total credit-limit for all the customers in the customer table. Each time the value of the state field changes, Progress displays a subtotal of the credit-limit values for customers in that state.

You can use the BREAK option anywhere in the block header, but you must also use the BY option to name a sort field.

You can use the BREAK option in conjunction with the ACCUMULATE statement and ACCUM function. For more information, see the reference entries for those language elements.

BY *expression* [DESCENDING]

Sorts the selected records by the value of *expression*. If you do not use the BY option, Progress retrieves records in the order of the index used to satisfy the *record-phrase* criteria, or the primary index if no criteria is given. The DESCENDING option sorts the records in descending order (not in the default ascending order).

Note: You cannot reference a BLOB or CLOB field in the BY option.

You can use multiple BY options to do multi-level sorting. For example:

```
FOR EACH customer BY credit-limit BY name
```

Here, the customers are sorted in order by credit-limit. Within each credit-limit value, customers are sorted alphabetically by name.

There is a performance benefit if an index on expression exists: BREAK BY does not have to perform the sort that is otherwise required to evaluate FIRST, LAST, FIRST-OF, and LAST-OF expressions.

COLLATE (*string* , *strength* [, *collation*]) [DESCENDING]

Generates the collation value of a string after applying a particular strength, and optionally, a particular collation. The DESCENDING option sorts the records in descending order (not in default ascending order).

string

A CHARACTER expression that evaluates to the string whose collation value you want to generate.

strength

A CHARACTER expression that evaluates to a Progress comparison strength or an International Components for Unicode (ICU) comparison strength.

The Progress comparison strengths include:

RAW — Generates a collation value for the string based on its binary value.

CASE-SENSITIVE — Generates a case-sensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, Progress applies the ICU TERTIARY strength.

CASE-INSENSITIVE — Generates a case-insensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, Progress applies the ICU SECONDARY strength.

CAPS — Generates a collation value for the string based on its binary value after converting any lowercase letters in the string to uppercase letters, based on the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters.

The ICU comparison strengths include:

PRIMARY — Generates a collation value for the base characters in the string.

SECONDARY — Generates a collation value for the base characters and any diacritical marks in the string.

TERTIARY — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string.

QUATERNARY — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string, and distinguishes words with and without punctuation. ICU uses this strength to distinguish between Hiragana and Katakana when applied with the ICU-JA (Japanese) collation. Otherwise, it is the same as TERTIARY.

IGNORE-SECONDARY — Generates a case-sensitive, but diacritically-insensitive, collation value for the string.

Note: Use ICU comparison strengths only with ICU collations.

collation

A CHARACTER expression that evaluates to the name of a Progress collation table or ICU collation. If *collation* does not appear, COLLATE uses the collation table of the client.

Progress reports an error and stops execution if one of the following occurs:

- *strength* does not evaluate to a valid value.
- *collation* does not evaluate to a collation table residing in the convmap.cp file.
- *collation* evaluates to a collation table that is not defined for the code page corresponding to the -cpinternal startup parameter.

variable = *expression1* TO *expression2* [BY *k*]

Identifies the name of a field or variable whose value you are incrementing in a loop. The *expression1* is the starting value for *variable* on the first iteration of the loop. The *k* is the amount to add to *variable* after each iteration and must be a constant. It (*k*) defaults to 1. The *variable, expression1*, and *expression2* parameters must be integers.

When *variable* exceeds *expression2* (or is less than *expression2* if *k* is negative) the loop ends. Since *expression1* is compared to *expression2* at the start of the first iteration of the block, the block can be executed 0 times. Progress re-evaluates *expression2* on each iteration of the block.

WHILE *expression*

Indicates the condition in which you want the FOR EACH block to continue processing the statements within it. Using the WHILE *expression* option causes the block to iterate as long as the condition specified by the expression is TRUE or Progress reaches the end of the index it is scanning, whichever comes first. The expression is any combination of constants, operators, field names, and variable names that yield a logical value.

TRANSACTION

Identifies the FOR EACH block as a system transaction block. Progress starts a system transaction for each iteration of a transaction block if there is not already an active system transaction. See *OpenEdge Development: Progress 4GL Handbook* for more information on transactions.

on-error-phrase

Describes the processing that takes place when there is an error during a block. This is the syntax for the ON ERROR phrase:

Syntax

```
ON ERROR UNDO  [ label1 ]
  [   , LEAVE  [ label2 ]
    | , NEXT   [ label2 ]
    | , RETRY  [ label1 ]
    | , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON ERROR phrase reference entry.

on-endkey-phrase

Describes the processing that takes place when the ENDKEY condition occurs during a block. This is the syntax for the ON ENDKEY phrase:

Syntax

```
ON ENDKEY UNDO [ label1 ]
  [   , LEAVE  [ label2 ]
    | , NEXT   [ label2 ]
    | , RETRY  [ label1 ]
    | , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON ENDKEY phrase reference entry.

on-quit-phrase

Describes the processing that takes place when a QUIT statement is executed during a block. This is the syntax for the ON QUIT phrase:

Syntax

```
ON QUIT [ UNDO [ label1 ] ]
  [   , LEAVE  [ label2 ]
    | , NEXT   [ label2 ]
    | , RETRY  [ label1 ]
    | , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON QUIT phrase reference entry.

on-stop-phrase

Describes the processing that takes place when the STOP conditions occurs during a block. This is the syntax for the ON STOP phrase:

Syntax

```
ON STOP UNDO [ label1 ]
  [   , LEAVE  [ label2 ]
    | , NEXT   [ label2 ]
    | , RETRY  [ label1 ]
    | , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON STOP phrase reference entry.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Examples

This procedure reads customer records that have a cust-num less than 12, sorting the records in order by state before displaying them:

r-fore.p

```
FOR EACH customer WHERE cust-num < 12 BY state:
     DISPLAY cust-num name city state.
END.
```

The next procedure gets information from four related tables (customer, order, order-line, and item) and displays some information from each. Before displaying the information, the FOR EACH statement sorts it in order by the promise-date field, then, within that field, in order by cust-num. Within the cust-num field, the data is sorted by the line-num field.

r-fore2.p

```
FOR EACH customer, EACH order OF customer,
    EACH order-line OF order, item OF order-line
    BY promise-date BY customer.cust-num BY line-num:
         DISPLAY promise-date customer.cust-num
                 order.order-num line-num item.item-num item-name.
END.
```

This procedure uses the LAST option to display information on the last order of each customer:

r-fore3.p

```
FOR EACH customer, LAST order OF customer:
     DISPLAY customer.cust-num customer.name order.order-num
             order.order-date order.instructions.
     PAUSE 1 NO-MESSAGE.
     instructions = "Last order".
     DISPLAY instruction.
END.
```

Notes

- At compile time, Progress determines which index or indexes to use for retrieving records from a table, based on the conditions in the Record phrase. For compatibility with Progress Version 6 or earlier, you can force Progress to use only one index by specifying the USE-INDEX option or by using the Version 6 Query (-v6q) parameter.
- If you specify the -v6q startup parameter, an index component is involved in an equality match if it is used in the Record phrase conditions in the following form:

Syntax

```
field = expression
```

Where the *expression* is independent of any fields in the table that the index is being selected from. A condition involving OF and USING are equivalent to this form. A field is involved in a range match if it is used in a condition of this form:

Syntax

```
field [ < | <= | > | >= | BEGINS ] expression
```

Note: The BEGINS operator translates into two range matches for a field.

An equality or range match is considered active if the equality or range condition stands on its own or is related to other conditions solely through the AND operator (for example, not through OR or NOT).

A field is involved in a sort match if it is used in a BY option of this form:

Syntax

```
BY field [ DESCENDING ]
```

- If you specify the -v6q startup parameter, the following list describes the rules the OpenEdge database manager uses to choose an index for an OpenEdge database:
  - If you specify the record by ROWID, Progress accesses the record directly without using an index.
  - If you use the USE-INDEX option, in the *record-phrase*, Progress uses the index you name in that option.
  - For each index in the table, Progress looks at each index component in turn and counts the number of active equality, range, and sort matches. Progress ignores the counts for any components of an index that occur after a component that has no active equality match. Progress compares the results of this count and selects the best index. Progress uses the following order to determine the better of any two indexes.
    1. If one index is unique and all of its components are involved in active equality matches and the other index is not unique, or if not all of its components are involved in active equality matches, Progress chooses the former of the two.
    2. Select the index with more active equality matches.
    3. Select the index with more active range matches.
    4. Select the index with more active sort matches.
    5. Select the index that is the primary index.
    6. Select the first index alphabetically by index name.
- If you specify the -v6q startup parameter, Progress might have to scan all the records in the index to find those meeting the conditions, or Progress might have to examine only a subset of the records. This latter case is called bracketing the index and results in more efficient access. Having selected an index as previously described, Progress examines each component as follows to see if the index can be bracketed:
  - If the component has an active equality match, Progress can bracket it, and it examines the next component for possible bracketing.
  - If the component has an active range match, Progress can bracket it, but it does not examine the remaining components for possible bracketing.
  - If the component does not have an active equality match or an active range match, Progress does not examine the remaining components for bracketing.

- If you specify the v6q parameter, any conditions you specify in the *record-phrase* that are not involved in bracketing the selected index are applied to the fields in the record itself to determine if the record meets the overall *record-phrase* criteria. For example, assume that the f table has fields a, b, and c involved in two indexes:

  – Primary, unique index (I1) on a, b, and c.

  – Secondary non-unique index (I2) on c.

  Table 31 shows the index Progress selects and the bracketed part of the index for various *record-phrases*.

Table 31: Progress version 6 index selection examples

Record phrase	Index selected	Bracketing on
f WHERE a = 3 AND b = 2 AND c = 3	I1	a + b + c
f WHERE a = 3	I1	a
f WHERE c = 1	I2	c
f WHERE a = 3 AND b > 7 AND c = 3	I1	a + b
f WHERE a = 3 AND c = 4	I1	a
f WHERE b = 5	I1	None of the fields[1]
f WHERE a = 1 OR b >5	I1	None of the fields[1]
f WHERE (a >= a1 AND a <= a2) OR (a1=0)	I1	None of the fields[2]
f WHERE a >= (IF a1 NE 0 THEN a1 ELSE -99999999) AND a <= (IF a1 NE 0 THEN a2 ELSE +99999999)	I1	a[2]

[1] In this case, Progress must look at all of the records to determine which meet the specified criteria.

[2] The two record phrases in these examples are almost identical in effect, but the one using the OR operator to connect conditions is much less efficient in its use of the selected index.

- The FIRST and LAST keywords are especially useful when you are sorting records in a table in which you want to display information. Often, several related records exist in a related table, but you only want to display the first or last related record from that table in the sort. You can use FIRST or LAST in these cases. Two examples follow.

  Suppose you were interested in displaying the date when each customer first placed an order. This procedure displays the customer number and date of the first order:

```
FOR EACH customer, FIRST order OF customer:
  DISPLAY order.cust-num order-date.
END.
```

  The following procedure displays the last order line of every order, sorted by the price of the item and by the promised date of the order:

```
DISPLAY "Show the last order-line of each order," SKIP
      sorted by the item's price and the" SKIP
      promised date of the order." WITH CENTERED.

FOR EACH order, LAST order-line OF order,
         item OF order-line BY item.price BY order.promise-date:
   DISPLAY order.order-num line-num item.item-num price
           promise-date WITH TITLE "For FIRST/LAST" CENTERED.
END.
```

- If you want Progress to use a specific index, you must specify the first component of that index in the record phrase of the FOR statement.
- You cannot reference a BLOB or CLOB field in a WHERE clause.
- For more information on the FOR statement, see *OpenEdge Development: Progress 4GL Handbook*.
- For SpeedScript, the *on-endkey-phrase* and the *on-quit-phrase* do not apply.

See also FIND statement, Frame phrase, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, ON STOP phrase, Record phrase

FORM statement

Defines the layout and certain processing attributes of a frame for use within a single procedure. If the frame has not been previously scoped, the FORM statement scopes it to the current block. Use the FORM statement if you want to describe a frame in a single statement rather than let Progress construct the frame based on individual data handling statements in a block. You can use the FORM statement to describe a layout for a data iteration and the frame header or background.

Syntax

```
FORM
  [ form-item  ... ]
  [ { HEADER | BACKGROUND } head-item ... ]
  [ frame-phrase ]
```

```
FORM record [ EXCEPT field ... ] [ frame-phrase ]
```

form-item

Specifies a field-level widget or value to display in the frame, or a SPACE or SKIP directive. The data specified by all form items are owned by a single field group, duplicated for each data iteration in the frame.

This is the syntax for *form-item*:

Syntax

```
field [ format-phrase ]
```

```
constant
  [ at-phrase | TO n ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ FGCOLOR expression ]
  [ FONT expression ]
  [ PFCOLOR expression ]
  [ VIEW-AS TEXT ]
  [ WIDGET-ID id-number ]
```

```
SPACE [ ( n ) ]
```

```
SKIP [ ( n ) ]
```

field

A reference to a field or variable to be displayed in the frame. This value cannot be an expression or a frame. To specify a child frame, you must first define the parent and child frames, then assign the FRAME attribute of the child frame to the widget handle of the parent frame. The child frame is assigned to the same field group as other form items.

format-phrase

Specifies one or more frame attributes for a field or variable. For more information on *format-phrase*, see the Format phrase reference entry.

constant

A constant value.

at-phrase

Specifies the location of a value within the frame. The AT phrase does not left justify the data; it simply indicates the placement of the data area. This is the syntax for the AT phrase:

Syntax

```
AT { n
   | { COLUMN column | COLUMN-OF relative-position }
     { ROW row | ROW-OF relative-position }
     [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
   | { X x | X-OF relative-position }
     { Y y | Y-OF relative-position }
     [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
   }
```

For more information, see the AT phrase reference entry.

TO *n*

The number (*n*) of the column in which you want the display to end. The TO option does not right justify the data; it simply indicates the placement of the data area.

BGCOLOR *expression*

Specifies the background color of the form item in graphical interfaces. This option is ignored in character interfaces.

DCOLOR *expression*

Specifies the display color of the form item in character interfaces. This option is ignored in graphical interfaces.

FGCOLOR *expression*

Specifies the foreground color of the form item in graphical interfaces. This option is ignored in character interfaces.

FONT *expression*

Specifies the font of the form item.

PFCOLOR *expression*

Specifies the prompt color of the form item in character interfaces. This option is ignored in graphical interfaces.

VIEW-AS TEXT

Specifies that the form item be displayed as a TEXT widget rather than as a FILL-IN widget.

WIDGET-ID *id-number*

Specifies a widget ID for a field-level widget or value to display in a frame. The value of *id-number* must be an expression that evaluates to an even integer value between 2 and 65534, inclusive, and must be unique across all widget IDs in the window or dialog box.

If you specify an invalid ID, the compiler displays an error message. This option is supported in graphical interfaces only, and only in Windows.

SPACE (*n*)

Identifies the number (*n*) of blank spaces to insert after the displayed expression. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, Progress starts a new line and discards extra spaces. If you do not use this option or you do not use *n*, Progress inserts one space between items in the frame.

SKIP (*n*)

Identifies the number (*n*) of blank lines to insert after the displayed expression. The number of blank lines can be can be 0. If you do not use this option, Progress does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option but do not specify *n*, or if *n* is 0, Progress starts a new line unless it is already at the beginning of a new line.

record

Represents the name of the record you want to display. Naming a record is shorthand for listing each field individually, as a form item.

EXCEPT *field* . . .

Tells Progress to display all the fields in the frame except those fields listed in the EXCEPT phrase.

HEADER

Tells Progress to place the following items in a header section at the top of the frame in a separate field group from all other data. In addition to fields, variables, and constants, the frame header can contain expressions, images, and rectangles. Progress reevaluates these expressions each time it displays the frame.

When you use the FORM statement with the HEADER option, Progress disregards Data Dictionary field labels for fields you name in the FORM statement. Use character strings to specify labels for fields you name in the frame header.

BACKGROUND

Specifies that any following frame items display in the frame background, behind the data and header in a separate field group. Typically, this option is used to display images or rectangles behind the data.

head-item

A description of a value to be displayed in the frame header or background, or a SPACE or SKIP directive. This is the syntax for head-item:

Syntax

```
expression [ format-phrase ]
```

```
constant
  [ at-phrase | TO n ]
  [ BGCOLOR expression ]
  [ DCOLOR expression ]
  [ FGCOLOR expression ]
  [ FONT expression ]
  [ VIEW-AS TEXT ]
  [ WIDGET-ID id-number ]
```

```
SPACE [ ( n ) ]
```

```
SKIP [ ( n ) ]
```

This is exactly the same as the syntax for a *form-item*, except that a *head-item* can be an expression and does not include the PFCOLOR option. If you use an expression in a HEADER or BACKGROUND phrase, the expression is evaluated each time the frame is viewed. If you give the PAGE-TOP or PAGE-BOTTOM option for the frame, the expression is evaluated for each page. This allows you, for example, to include a reference to the PAGE-NUMBER function in the frame header.

Note: If *head-item* is an expression, any option of the *format-phrase* may be used with it; if *head-item* is a constant, only the AT phrase, TO, BGCOLOR, DCOLOR, FGCOLOR, FONT, VIEW-AS TEXT, and WIDGET-ID options are allowed.

frame-phrase

Specifies frame options for the frame associated with the FORM statement. For more information on *frame-phrase* options, see the Frame phrase reference entry.

Examples

This procedure lets the user update information on a specific customer. The FORM statement describes a very specific layout for the UPDATE statement to use.

r-form.p

```
REPEAT FOR customer:
    FORM name    COLON 10         phone     COLON 50
         address COLON 10         sales-rep COLON 50  SKIP
         city    COLON 10 NO-LABEL state NO-LABEL postal-code NO-LABEL
        WITH SIDE-LABELS 1 DOWN CENTERED.
    PROMPT-FOR cust-num WITH FRAME cnum SIDE-LABELS CENTERED.
    FIND customer USING cust-num.
    UPDATE name address city state postal-code phone sales-rep.
END.
```

When you use the FORM statement to control the order in which fields appear on the screen, remember that this order is independent of the order in which Progress processes the fields during data entry.

In the example, the above FORM statement displays the customer name first and the phone number second. But the UPDATE statement specifies the phone number after the name, address, city, state, and postal-code. The fields are displayed as described in the FORM statement, but the tab order is determined by the UPDATE statement.

The following example uses the HEADER option:

r-eval.p

```
DEFINE VARIABLE i AS INTEGER FORMAT ">9".
FORM HEADER "This is the header - i is" i
    WITH FRAME a ROW i COLUMN i i DOWN.
DO i = 1 TO 8 WITH FRAME a.
  DISPLAY i.
  PAUSE.
END.
```

The FORM statement defines a HEADER frame that consists of the text “This is the header - i is” and the value of the variable i. In addition, it also specifies a screen location where the header is displayed. The FORM statement does not bring the header frame into view.

On the first iteration of the DO block, the DISPLAY statement brings the frame into view. On the second iteration of the DO block, the frame is already in view (it was not hidden during the first iteration), so the header of the frame is not re-evaluated. Thus, the new value of i is not reflected in the header portion of the frame, and you do not see the new value of i in the header. You also do not see the position of the frame on the screen change.

In contrast, look at this modified version of the procedure:

r-eval2.p

```
DEFINE VARIABLE i AS INTEGER FORMAT ">9".
FORM HEADER "This is the header - i is" i
    WITH FRAME a ROW i COLUMN i i DOWN.
DO i = 1 TO 8 WITH FRAME a.
  DISPLAY i.
  HIDE FRAME a.
END.
```

On the first iteration of the DO block, the DISPLAY statement displays the frame. The HIDE statement removes the frame from the window. Therefore, on the second iteration of the DO block, the DISPLAY statement redisplays the frame. Progress re-evaluates the header of the frame each time the frame is redisplayed. Therefore, the header of the frame reflects the change to i, and the position of the frame in the window also changes.

Notes

- When you use any of the statements that access the screen, you can name a frame or use the default frame for the block where the statements appears. For more information on frame scoping, see *OpenEdge Development: Progress 4GL Handbook*.
- When Progress compiles a procedure, it makes a top-to-bottom pass of the procedure to design all the frames for that procedure, including those referenced in FORM statements. Progress adds field and format attributes as it goes through the procedure.
- If you have enabled application-defined widget IDs in your OpenEdge GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then Progress uses the value specified in the WIDGET-ID option to set the WIDGET-ID attribute for this widget when it creates the widget at runtime, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then Progress ignores this option setting at runtime.

  For more information about the WIDGET-ID attribute, see its reference entry in the "Attributes and Methods Reference" section on page 1497. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- If you use a single qualified identifier with the FORM statement, the compiler first interprets the reference as *`dbname.tablename`*. If the compiler cannot resolve the reference as *`dbname.tablename`*, it tries to resolve it as *`tablename.fieldname`*. When naming fields in a FORM statement, you must use table names that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.
- To use the FORM statement to display a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.
- If you define a frame to use as a DDE frame, you must realize the frame (display it) before using it as a conversation end-point. If you want the DDE frame to remain invisible during its use in a DDE conversation, set its HIDDEN attribute to TRUE after realizing the frame. For information on DDE frames, see *OpenEdge Development: Programming Interfaces*.

See also DEFINE FRAME statement, Format phrase, Frame phrase

Format phrase

Specifies one or more attributes for a widget.

Syntax

```
[ at-phrase ]
[ AS datatype | LIKE field ]
[ ATTR-SPACE | NO-ATTR-SPACE ]
[ AUTO-RETURN ]
[ BGCOLOR expression ]
[ BLANK ]
[ COLON n | TO n ]
[ COLUMN-LABEL label  ]
[ DEBLANK ]
[ DCOLOR expression ]
[ DISABLE-AUTO-ZAP ]
[ FGCOLOR expression ]
[ FONT expression ]
[ FORMAT expression ]
[ HELP string ]
[ LABEL label [ , label ] ... | NO-LABELS ]
[ NO-TAB-STOP ]
[ PFCOLOR expression ]
[ VALIDATE ( condition , msg-expression ) ]
[ view-as-phrase ]
[ WIDGET-ID id-number ]
```

at-phrase

The column, row and column, or x and y pixel location you want the display to start. The AT option does not left justify the data; it simply indicates the placement of the data area.

Syntax

```
AT { n
   | { COLUMN column | COLUMN-OF relative-position }
     { ROW row | ROW-OF relative-position }
     [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
   | { X x | X-OF relative-position }
     { Y y | Y-OF relative-position }
     [ COLON-ALIGNED | LEFT-ALIGNED | RIGHT-ALIGNED ]
   }
```

See the AT phrase reference entry for more information.

AS *datatype*

Creates a frame field and variable with the data type you specify. This is useful for defining display positions in a frame for use with DISPLAY @ *field*.

LIKE *field*

Creates a frame field and variable with the same definition as *field*.

The LIKE option in a DEFINE VARIABLE statement, DEFINE WORK-TABLE statement, or Format phrase requires that a particular database is connected. Since you can start up an OpenEdge application session without connecting to a database, use the LIKE option with caution.

ATTR-SPACE | NO-ATTR-SPACE

Has no effect; supported only for backward compatibility.

AUTO-RETURN

Causes Progress to automatically move out of a field as if you pressed **RETURN**. When you enter the last character in the field, Progress automatically moves out of the field. If this happens on the last field of a data entry statement, Progress functions as if you pressed **GO**.

For the purposes of AUTO-RETURN, entering leading zeros in a numeric field does not count as filling the field. For example, suppose you define a numeric field as follows:

```
DEFINE VARIABLE x AS INTEGER FORMAT "99".
SET x AUTO-RETURN.
```

If you enter a 09 into the field, Progress does not AUTO-RETURN. To get the AUTO-RETURN behavior in this situation, define the field as CHARACTER with a format of "99".

BGCOLOR *expression*

Specifies the background color of the widget in graphical interfaces. This option is ignored in character interfaces.

BLANK

Displays blanks for the field you are displaying or entering. This is useful for entering passwords.

COLON *n*

The number (*n*) of the column in which you want the colon of the label to appear. Use this option with SIDE-LABEL frames where the labels are placed to the left of the data and are separated from the data with a colon.

TO *n*

The number (*n*) of the column in which you want to end the display. The TO option does not right justify the data; it indicates the placement of the data area.

COLUMN-LABEL *label*

Names the label you want to display above the field. If you want the label to use more than one line (stacked labels), use an exclamation point (!) in the label to indicate where to break the line.

r-colbl.p

```
FOR EACH customer:
  DISPLAY name COLUMN-LABEL "Customer!Name"
    sales-rep COLUMN-LABEL "Name of!Sales!Representative".
END.
```

Progress does not display column labels if you use the SIDE-LABELS or the NO-LABELS option with the Frame phrase.

You must enclose the label string in quotation marks. If you want to use the exclamation point (!) as one of the characters in a column label, use two exclamation points (!!).

DEBLANK

Removes leading blanks (for use on input character fields only). Leading blanks in the value before input are not removed unless the user changes the value.

DCOLOR *expression*

Specifies the display color of the widget in character interfaces. This attribute is ignored in graphical interfaces.

DISABLE-AUTO-ZAP

Specifies whether the value of the AUTO-ZAP attribute will be ignored. See the AUTO-ZAP attribute reference entry. This option only applies to fill-ins.

The following example defines a frame with two fill-ins, both of which specify the DISABLE-AUTO-ZAP option:

```
DEFINE FRAME frame-a
  fill-in-1 DISABLE-AUTO-ZAP
  fill-in-2 DISABLE-AUTO-ZAP
  button-1
  WITH THREE-D SIDE-LABELS.
```

`FGCOLOR` *`expression`*

Specifies the foreground color of the widget in graphical interfaces. This option is ignored in character interfaces.

`FONT` *`expression`*

Specifies the font of the widget.

`FORMAT` *`string`*

Represents the format in which you want to display the *`expression`*. You must enclose string in quotation marks (""). If you do not use the FORMAT option, Progress uses the defaults shown in Table 32.

Table 32: Default display formats

Type of expression	Default format
Field	Format from Dictionary.
Variable	Format from variable definition.
Constant character	Length of character string.
Other	Default format for the data type of the expression.

Table 33 lists the default formats for the Other expression.

Table 33: Default data type display formats *(1 of 2)*

Data type	Default display format
CHARACTER	x(8)
DATE	99/99/99
DATETIME	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	->>,>>9.99
HANDLE[2]	>>>>>>9

Table 33: Default data type display formats *(2 of 2)*

Data type	Default display format
INTEGER	->,>>>,>>9
LOGICAL	yes/no
MEMPTR[1]	See the footnote at the end of this table.
RAW[1]	See the footnote at the end of this table.
RECID	>>>>>>9
ROWID[1]	See the footnote at the end of this table.
WIDGET-HANDLE[2]	>>>>>>9

[1] You cannot display a MEMPTR, RAW, or ROWID value directly. However, you can convert it to a character string representation using the STRING function and display the result. A ROWID value converts to a hexadecimal string, "0x*hexdigits*," where *hexdigits* is any number of characters "0" through "9" and "A" through "F". A MEMPTR or RAW value converts to decimal integer string.

[2] To display a HANDLE or WIDGET-HANDLE, you must first convert it using the INTEGER function and display the result.

You can use the FORMAT option with the UPDATE and SET statements to store a character string that is longer than the field length you define in the Data Dictionary or in a DEFINE VARIABLE statement. This is possible because Progress stores data in variable-length fields.

```
DEFINE VARIABLE mychar AS CHARACTER FORMAT "x(3)".
UPDATE mychar FORMAT "x(8)".
```

You can also use the ASSIGN statement to store data in a field or variable that is longer than the predefined format of that field or variable.

```
mychar = "abcdefgh".
```

However, the Data Dictionary load program only loads character data that is **no longer** than the format you defined in the Dictionary. For more information on data formats, see *OpenEdge Development: Progress 4GL Handbook*.

HELP *string*

Represents a character string that you want to display whenever the user enters the frame field for the field or variable. When the user leaves the frame field, Progress removes the help string from the message area. You must enclose the string in quotation marks ("").

If the input source is not the terminal, Progress disregards any HELP options.

LABEL *label* **[** , *label* **]** **. . .**

Represents a character string that you want to use as a label for a field, variable, or expression. You must enclose the string in quotation marks (""). Table 34 shows the order Progress uses to determine the label for a field, variable, or expression.

Table 34: Determining labels

	LABEL *string*	Dictionary label	Field name	LIKE *field*	Variable name
Field	1	2	3	N/A	N/A
Variable	1	N/A	N/A	2	3
Expression	1	N/A	N/A	N/A	N/A

Note: If you use side labels, Windows allows a user to transfer focus to field-level widgets by pressing **ALT** and one of the letters in the widget's label. This is called a mnemonic. Specify the letter by preceding it with an ampersand (&) when specifying the LABEL option. Ending a label with an ampersand might produce undesired behavior. If you want a literal ampersand within a label, enter two ampersands (&&) in *label*. If you specify more than one widget with the same mnemonic, Progress transfers focus to each of these in tab order when you make a selection.

NO-LABELS

Prevents Progress from displaying a label for a field, variable, or expression.

NO-TAB-STOP

Specifies that the widget is not in its parent frame's tab order.

The following example shows defining a frame with two fill-ins, both of which have the NO-TAB-STOP option specified:

```
DEFINE FRAME frame-a
  fill-in-1
  fill-in-2
  button-1 NO-TAB-STOP
  WITH THREE-D SIDE-LABELS.
```

See the TAB-STOP attribute reference entry for related information.

PFCOLOR *expression*

Specifies the prompt color of the widget in character interfaces. This attribute is ignored in graphical interfaces.

VALIDATE (*condition*, *msg-expression*)

Specifies a value that you want to validate against the data entered into a screen field or variable. The *condition* is a Boolean expression (a constant, field name, variable name, or expression) whose value is TRUE or FALSE.

When you use the VALIDATE option to validate a specific field, any reference to that field in *condition* is assumed to be an input field. For example, in the following statement, Progress assumes the promise-date field is an input field.

```
SET order-date promise-date VALIDATE(promise-date > order-date,
  "Promise date must be later than order date").
```

The previous statement is equivalent to the following statement:

```
SET order-date promise-date VALIDATE(INPUT promise-date > order-date,
  "Promise date must be later than order date").
```

The validation is based on the value of order-date prior to the SET statement. If you want to validate the value of promise-date against the input value of order-date, use this statement:

```
SET order-date promise-date VALIDATE(promise-date > INPUT order-date,
  "Promise date must be later than order date").
```

If you try to validate a field whose reference is ambiguous, Progress tries to resolve the ambiguity by referencing the table that contains the record being updated. In the following example, the sales-rep field is ambiguous because it exists in both the order table and the customer table. Progress resolves the ambiguity by validating the sales-rep field in the order table, since the order table is being updated.

```
FIND FIRST customer.
FIND FIRST order.
UPDATE order.cust-num order.sales-rep
  VALIDATE(LENGTH(sales-rep) > 1,
           "Invalid sales-rep value.").
```

If the reference is to an array field and has no subscript, Progress assumes you want to use the subscript of the field that is being prompted.

If the value of *condition* is FALSE, use *msg-expression* to display a specific message. You must enclose *msg-expression* in quotation marks (" ").

Progress processes validation criteria whenever the user attempts to leave the frame field. If the frame field value is not valid, Progress displays *msg-expression* in the message area, causes the terminal to beep, and does not advance out of the frame field.

If you tab a frame field, make no changes, and leave the field, Progress does not process the validation criteria specified with the VALIDATE option until the you press **GO** (**F1**). If you press **ENDKEY** or **END-ERROR**, or an error occurs, Progress does not test the validation criteria specified with the VALIDATE option.

If the input source for the procedure is a table, Progress validates each input field (except those with a value of "-"). If the result of the validation is FALSE, *msg-expression* is displayed and Progress treats the validation as an error.

To suppress the Data Dictionary validation criteria for a field, use this VALIDATE option.

```
VALIDATE(TRUE,"")
```

When you use the VALIDATE option in a procedure to specify validation criteria for a field, that validation criteria applies to all other references to that field in the same frame.

```
FOR EACH order:
  UPDATE order-date.
  UPDATE order-date VALIDATE(order-date LE TODAY,
                             "Can't be later than today").
END.
```

In this example, Progress applies the validation criteria on the second UPDATE statement. Progress also applies the validation criteria to the first UPDATE statement because both UPDATE statements use the same frame. Scope references to the same field to different frames if you do not want a VALIDATE option to affect all references to that field.

view-as-phrase

Specifies the type of widget. This is the syntax for view-as-phrase:

Syntax

```
VIEW-AS {    editor-phrase
           | FILL-IN [ NATIVE ] [ size-phrase ]
           | radio-set-phrase
           | selection-list-phrase
           | slider-phrase
           | TEXT [ size-phrase ]
           | TOGGLE-BOX [ size-phrase ]
         }
```

For more information on *view-as-phrase*, see the VIEW-AS phrase reference entry.

WIDGET-ID *id-number*

Specifies a widget ID for a field or variable widget to display in a frame. The value of *id-number* must be an expression that evaluates to an even integer value between 2 and 65534, inclusive, and must be unique across all widget IDs in the window or dialog box.

If you specify an invalid ID, the compiler displays an error message. This option is supported in graphical interfaces only, and only in Windows.

Example This procedure lets the user update customer records after entering the password "secret." The format phrase on the phone field describes the display format of that field.

r-frmat.p

```
DEFINE VARIABLE password AS CHARACTER.

UPDATE password FORMAT "x(6)" BLANK
    VALIDATE(password = "secret", "Sorry, wrong password")
    HELP "Maybe the password is 'secret' !"
    WITH FRAME passw CENTERED SIDE-LABELS.
HIDE FRAME passw.

REPEAT:
    PROMPT-FOR customer.cust-num COLON 20.
    FIND customer USING cust-num.
    UPDATE
    name     LABEL "Customer Name" COLON 20
             VALIDATE(name ne "", "Please enter a name")
    address  HELP  "Please enter two lines of address"
             COLON 20 LABEL "Address"
    address2 NO-LABEL COLON 20
    city     COLON 20
    state    COLON 20
    postal-code    COLON 20  SKIP(3)
    phone    AT 5 FORMAT "(999) 999-9999"
    contact  TO 60
    WITH CENTERED SIDE-LABELS.
END.
```

Notes

- The ATTR-SPACE/NO-ATTR-SPACE designation in a Frame phrase takes precedence over an ATTR-SPACE/NO-ATTR-SPACE designation in a Format phrase. The ATTR-SPACE/NO-ATTR-SPACE designation in a Format phrase takes precedence over an ATTR-SPACE/NO-ATTR-SPACE designation in a COMPILE statement.
- If you have enabled application-defined widget IDs in your OpenEdge GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then Progress uses the value specified in the WIDGET-ID option to set the WIDGET-ID attribute for this widget when it creates the widget at runtime, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then Progress ignores this option setting at runtime.

  For more information about the WIDGET-ID attribute, see its reference entry in the "Attributes and Methods Reference" section on page 1497. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- For SpeedScript, these options are invalid: BGCOLOR, DCOLOR, FGCOLOR, FONT, PFCOLOR, *`view-as phrase`*.
- With respect to internationalization, some double-byte and UTF-8 multi-byte characters display and print in one or two columns. Each unit in the format string represents one physical column. To display or print a character that requires two columns, the FORMAT phrase must specify two columns. For more information, see *OpenEdge Development: Internationalizing Applications*.

See also FORM statement, Frame phrase

Frame phrase

Specifies the overall layout and processing properties of a frame for frame definition (DEFINE FRAME and FORM), block header (DO, FOR EACH, and REPEAT), and data handling (DISPLAY, SET, etc.) statements. When used on block header statements, the Frame phrase also specifies the default frame for data handling statements within the block. Frame phrases can also be used on individual data handling statements to indicate the specific frame where the statement applies.

Syntax

```
WITH [ ACCUM [ max-length ] ]
     [ at-phrase ] [ ATTR-SPACE | NO-ATTR-SPACE ]
     [ CANCEL-BUTTON button-name ] [ CENTERED ]
     [ color-specification ]
     [ COLUMN expression ] [ n COLUMNS ]
     [ CONTEXT-HELP ] [ CONTEXT-HELP-FILE help-file-name ]
     [ DEFAULT-BUTTON button-name ]
     [ DROP-TARGET ]
     [ [ expression ] DOWN ] [ EXPORT ]
     [ WIDGET-ID id-number ] [ FONT expression ]
     [ FRAME frame ] [ KEEP-TAB-ORDER ] [ NO-BOX ]
     [ NO-HIDE ] [ NO-LABELS ] [ USE-DICT-EXPS ]
     [ NO-VALIDATE ] [ NO-AUTO-VALIDATE ]
     [ NO-HELP ] [ NO-UNDERLINE ]
     [ OVERLAY ] [ PAGE-BOTTOM | PAGE-TOP ] [ RETAIN n ]
     [ ROW expression ] [ SCREEN-IO | STREAM-IO ]
     [ SCROLL n ] [ SCROLLABLE ] [ SIDE-LABELS ]
     [ size-phrase ] [ STREAM stream ] [ THREE-D ]
     [ title-phrase ] [ TOP-ONLY ] [ USE-TEXT ]
     [ V6FRAME [ USE-REVVIDEO | USE-UNDERLINE ] ]
     [ VIEW-AS DIALOG-BOX ] [ WIDTH n ] [ IN WINDOW window ]
```

ACCUM [*max-length*]

The ACCUM option lets you use aggregate functions (such as MAX, MIN, TOTAL, and SUBTOTAL) to accumulate values within shared frames. With the ACCUM option, aggregate values can be shared among procedures through shared frames. You must include the ACCUM option in the FORM statement or DEFINE FRAME statement of each procedure that uses the shared frame, as shown in the following examples:

```
DEFINE NEW SHARED FRAME x.
FORM field1 field2 WITH FRAME x ACCUM.
RUN testb.p.
```

```
/* testb.p */
DEFINE SHARED FRAME x.
FORM field1 field2 WITH FRAME x ACCUM.
FOR EACH table1:
 DISPLAY field1 field2 (TOTAL) WITH FRAME x.
END.
```

When you specify a user-defined aggregate label, use the *max-length* parameter of the ACCUM option to specify a maximum aggregate label length in the frame phrases of shared frames. For more information, see the Aggregate phrase reference entry.

at-phrase

Specifies the position of the frame (upper-left corner) within a window or parent frame. This is the syntax for the AT phrase for a frame:

Syntax

```
AT { COLUMN column ROW row
   | X x Y y
   }
```

Note that for a frame parented by a window, you must specify an absolute position relative to the display area of the window. For a frame parented by another frame, you must specify a position relative to the display area of the parent frame. The default value for all AT phrase parameters is 1. Progress ignores the COLUMN or X option if you use the CENTERED option for the same frame. For more information on *at-phrase*, see the AT phrase reference entry.

ATTR-SPACE | NO-ATTR-SPACE

Has no effect; supported only for backward compatibility.

CANCEL-BUTTON *button-name*

Specifies the cancel button for the frame. This is the button chosen when the **ESC** key code is applied to the frame in Windows. This button might also be chosen when the **ESC** key code is applied to a frame within the same frame family that does not have a cancel button. In such an event, Progress searches the frame family in random order. The first cancel button found during this random search is chosen. The *button-name* argument must be a static button name.

CENTERED

Centers the frame horizontally in the window or frame to which it is parented (or the terminal display, in character mode). If you use the CENTERED option and are sending output to a device other than the terminal, Progress centers the frame for the terminal. This might result in a non-centered frame on the alternate output device.

You can also use the AT phrase or COLUMN option to specify the position of the frame.

color-specification

For a graphical user interface, specifies the foreground and background color of the frame; for a character interface, specifies the display and prompt colors for the frame.

Syntax

```
[  { [ BGCOLOR expression ]
     [ DCOLOR expression ]
     [ FGCOLOR expression ]
     [ PFCOLOR expression ]
   }
   | { COLOR [ DISPLAY ] color-phrase
       [ PROMPT color-phrase ]
     }
]
```

For graphical interfaces, the FGCOLOR and BGCOLOR options specify the foreground and background color of the frame. These options are not supported in character interfaces. For character interfaces, use the DCOLOR and PFCOLOR options (which are not supported in graphical interfaces) to specify the display color and prompt color of the frame.

The COLOR option is obsolete, but is retained for backward compatibility.

Widgets (except child frames) within the frame inherit the colors of the frame by default. You can also set the colors of each widget individually.

COLUMN *expression*

The *expression* is a constant, field name, variable name or expression whose value is the number of the column, relative to the window or parent frame in which you place the frame. The default value is 1. Progress ignores this option if you use the CENTERED option for the same frame.

Progress evaluates *expression* each time the frame comes into view or is printed at the top or bottom of a page (if the frame is a PAGE-TOP or PAGE-BOTTOM frame). For more information, see the *expression* option of the FORM statement.

n COLUMNS

Formats data fields into a specific number (*n*) of columns. Truncates labels to 16, 14, and 12 characters when the number of columns is 1, 2, or 3, respectively. Progress reserves a fixed number of positions in each column for labels. For *n* = 1, 16 positions are allowed for a label; for *n* = 2, 14 positions are allowed; and for *n* = 3, 12 positions are allowed. Label positions include room for a colon and a space after the label. Labels are right justified if they are short, and truncated if they are too long. By default, Progress wraps fields across the frame for as many lines as required, placing labels above the fields.

When you use this option, it implies SIDE-LABELS and overrides any AT, COLON, TO, or SPACE options you might have used in the same Frame phrase.

CONTEXT-HELP

Specifies that context-sensitive help is available for this frame. This option is valid in Windows GUI only.

CONTEXT-HELP-FILE *help-file-name*

Specifies the complete path name of a help (.HLP) file associated with this frame. If CONTEXT-HELP-FILE is specified without CONTEXT-HELP, CONTEXT-HELP is assumed. This behavior can be overridden by setting the dialog box's CONTEXT-HELP attribute to FALSE at run time. This option is valid in Windows GUI only.

DEFAULT-BUTTON *button-name*

Specifies a default button for the frame. This is the button chosen when the ENTER key code in Windows is invoked for the frame. This button might also be chosen when the **ESC** key code is applied to a frame within the same frame family that does not have a default button. In such an event, Progress searches the frame family in random order. The first default button found during this random search is chosen. The *button-name* argument must be the name of a static button. This button must be defined with the DEFAULT option and cannot display an image.

DROP-TARGET

Indicates whether you want to be able to drop a file onto the object.

[*expression*] DOWN

Specifies that the frame is a down frame. A **down frame** is a frame that can display multiple occurrences of the set of fields defined in the frame. The *expression* is a constant, field name, variable name or expression whose value is the number of occurrences you want in the frame. If you specify 1 for *expression*, the frame is **not** a down frame.

Down frames are typically specified for iterative blocks. On the first iteration of the block, Progress displays the first set of data (a record, field, or variable value) as the first occurrence in the frame. After displaying the data, Progress advances to the next occurrence in the frame on the second iteration of the block, and displays the second set of data there. Progress continues advancing and displaying data for the number of occurrences specified by *expression*, and prompts to continue with another set of occurrences until all the data has been displayed. Progress evaluates *expression* each time the frame comes into view or is printed at the top or bottom of a page (if the frame is a PAGE-TOP or PAGE-BOTTOM frame). If you do not specify *expression*, Progress displays as many occurrences as can fit in the current window.

If you do not use the DOWN option, Progress automatically makes certain frames down frames, unless you specify otherwise (1 DOWN). For more information on frames and down frames, see *OpenEdge Development: Progress 4GL Handbook*.

EXPORT

This option is valid only for SQL.

WIDGET-ID *id-number*

Specifies a widget ID for a frame widget. The value of *id-number* must be an expression that evaluates to an even integer value between 2 and 65534, inclusive, and must be unique across all widget IDs in the window or dialog box.

If you specify an invalid ID, the compiler displays an error message. This option is supported in graphical interfaces only, and only in Windows.

FONT *expression*

Specifies the font of the frame. All widgets within a frame, except child frames, inherit the font of the frame by default. You can also set the font of each widget individually. By default, Progress uses the default system font.

FRAME *frame*

Defines new frames by giving them unique names. Whenever the same frame name is referred to in more than one Frame phrase, Progress combines the characteristics on each Frame phrase naming that frame. Progress also combines any frame characteristics used in data handling statements that name the same frame into the same frame description. This option is redundant for DEFINE FRAME statements. If you do not specify this option, Progress uses the default frame for the current block.

KEEP-TAB-ORDER

Prevents the frame-oriented I/O statements, ENABLE, UPDATE, SET, and PROMPT-FOR, from changing the tab order of your widgets in the frame. The tab order always remains the same as the order in which you first specify widgets in the frame. If you do not specify this option, Progress creates a new tab order based on the order specified in each frame-oriented I/O statement.

All attributes and methods that affect tab order (such as FIRST-TAB-ITEM and MOVE-AFTER-TAB), continue to change the tab order whether or not you specify this option. If you specify the option, these attributes and methods specify a new tab order for all frame-oriented I/O statements to follow.

NO-BOX

Does not display a box around the frame. If you do not use this option, Progress displays a box around the data you are displaying.

If you are sending data to a device other than a terminal and you do not use this option, Progress omits the sides and bottom line of the box and replaces the top line with blanks.

NO-HIDE

Suppress the automatic hiding of the frame (when the block where the frame is scoped iterates). The frame is hidden only if space is needed to display other frames.

NO-HIDE suppresses hiding for a frame only when the block where that frame is scoped iterates. For example:

```
FOR EACH customer:
  DISPLAY cust-num name.
  FOR EACH  order OF customer:
    DISPLAY order.order-num.
    DISPLAY "hello" WITH FRAME b COLUMN 60 NO-HIDE.
  END.
END.
```

In this example, Progress does not hide frame b when the inner block iterates. However, it does hide frame b when the outer block iterates. If you want the frame to stay in view during iterations of the outer block, scope the frame to that block.

NO-LABELS

Does not display labels. This option overrides any COLUMN-LABEL option you include in another phrase or statement.

NO-UNDERLINE

Does not underline labels appearing above fields.

USE-DICT-EXPS

Ensures that validation expressions and help strings from the Data Dictionary are compiled into the application. Typically, when the Progress compiler encounters a field reference in an input statement, Data Dictionary help and validation expressions are compiled in for that field, unless the field has a HELP or VALIDATE option (format phrase) attached in the input statement (or earlier in the procedure). In this case, the custom help or validation expression is used.

In Progress Version 7 and later, there are two syntax constructs that can enable a field for input without the compiler specifically knowing about it: ENABLE ALL and *widget-name*:SENSITIVE = YES.

When Progress encounters an ENABLE ALL statement, every field in the associated frame has Data Dictionary validation expressions and help strings compiled into the application. This closes any possible validation or help hole. As a side-effect, validation expressions and help strings that are not required might be compiled, but this will not affect the application.

This behavior places two important conditions on you. First, adding a field to a frame after the first ENABLE ALL is not desirable. Data Dictionary validation and help will not be compiled for this field. Second, any custom validation or help must come before the first ENABLE ALL. A good practice is to include these in the DEFINE FRAME or FORM statements.

In the case of *widget-name*:SENSITIVE = YES, there is more potential for validation and help holes. Since the compiler cannot predict whether these statements are used, in effect, as input statements, no help or validation is compiled. USE-DICT-EXPS explicitly compiles in all validation expressions and help strings for a frame. For each frame that you use *widget-name*:SENSITIVE = YES, specify USE-DICT-EXPS. This closes any potential validation or help holes. To provide custom help or validation when using USE-DICT-EXPS, the HELP or VALIDATE option must appear in the first reference to that field. Typically, this is in the DEFINE FRAME or FORM statement.

`NO-VALIDATE`

Disregards all validation conditions specified in the Data Dictionary for fields entered in this frame.

`NO-AUTO-VALIDATE`

Tells Progress to compile into the code all relevant validations it finds in the OpenEdge Data Dictionary, but to run the validations only when the code for the frame or for a field-level child-widget of the frame specifically invokes the VALIDATE() method.

`NO-HELP`

Disregards all help strings specified in the Data Dictionary for fields entered in this frame.

OVERLAY

Indicates that the frame can overlay any other frame that does not use the TOP-ONLY option. If you do not use this option, the frame you are using cannot overlay other frames. If Progress needs to display an OVERLAY frame and doing so will partially obscure a TOP-ONLY frame, it first hides the TOP-ONLY frame. Any frame parented by another frame is an OVERLAY frame within the parent frame.

This procedure uses the OVERLAY option on the Frame phrase:

r-ovrlay.p

```
FOR EACH customer:
  DISPLAY customer WITH 2 COLUMNS
    TITLE "Customer Information".
  FOR EACH order OF customer:
    DISPLAY order WITH 2 COLUMNS OVERLAY
    TITLE "Customer's Orders" ROW 7 COLUMN 10.
  END.
END.
```

The procedure above displays customer information in one frame. The procedure then displays order information for the customer in a second frame that overlays the first.

PAGE-BOTTOM

Displays the frame at the bottom of the page each time the output ends a page.

PAGE-TOP

Displays the frame each time the output begins on a new page.

Table 35 shows how the PAGE-TOP and PAGE-BOTTOM options work depending on the kind of DISPLAY or VIEW.

Table 35: Using PAGE-TOP and PAGE-BOTTOM frames

	OUTPUT to a PAGED file	OUTPUT to the screen or an UNPAGED file
DISPLAY a PAGE-TOP Frame	The frame is written to the file and put on a list of frames to be written at the top of each page.	The frame is written to the file or displayed on the screen.
VIEW a PAGE-TOP Frame	The frame is put on a list of frames to be written at the top of each page.	The frame is written to the file or displayed on the screen.
DISPLAY a PAGE-BOTTOM Frame	The frame is written to the file and put on a list of frames to be written at the bottom of each page.	The frame is written to the file or displayed on the screen.
VIEW a PAGE-BOTTOM Frame	The frame is put on a list of frames to be written at the bottom of each page.	The frame is written to the file or displayed on the screen.
HIDE Either Type	The frame is removed from the appropriate list.	The frame is removed from the screen.

`RETAIN` *n*

Specifies the number of frame iterations to retain when the frame scrolls on the screen. The *n* must be a constant. For example, RETAIN 2 causes Progress to display the last two iterations in a down frame at the top of the frame. If you are using UP to scroll up a window, those two lines are displayed at the bottom of the window. Do not use the SCROLL option in a Frame phrase in which you also use the RETAIN option. By default, Progress does not retain any iterations in the window that have already been displayed.

ROW *expression*

The *expression* is a constant, field name, variable name, function reference, or expression whose value is the row, relative to the window or parent frame in which you place the frame. If you are displaying a frame on a device other than a terminal, this option has no effect. By default, Progress displays a root frame at the next available row of the window and displays a child frame at row 1 of the parent frame.

Progress evaluates *expression* each time the frame comes into view or is printed at the top or bottom of a page (if the frame is a PAGE-TOP or PAGE-BOTTOM frame).

For more info, see the *expression* option of the FORM statement.

[SCREEN-IO | STREAM-IO]

If you specify STREAM-IO for a frame, the USE-TEXT option is assumed and all font specifications are ignored. The frame is formatted using a fixed font in a manner appropriate for streaming to a text file or printer. In particular, all border padding for FILL-IN widgets is dropped and the default system font is used.

If you use the STREAM-IO option on the COMPILE statement, this behavior is the default for all frames in the procedure. In this case, you can override that option by specifying SCREEN-IO for an individual frame.

SCROLL *n*

Displays a scrolling frame rather than a paging frame. The value *n* is a constant that specifies the number of frame iterations to scroll when the frame scrolls in the window. For example, if a procedure uses a DISPLAY or DOWN statement when a scrolling frame is full, the data in the frame scrolls up *n* iterations (rather than clearing and repainting the frame as it would without the SCROLL option).

This procedure uses the SCROLL option to scroll the display one line at a time:

r-fphrsc.p

```
FOR EACH customer WHERE cust-num <= 50:
   DISPLAY cust-num name credit-limit WITH SCROLL 1 USE-TEXT.
   IF credit-limit >= 50000
   THEN COLOR DISPLAY MESSAGES credit-limit.
END.
```

Do not use the RETAIN option in a Frame phrase in which you also use the SCROLL option.

SCROLLABLE

If you specify this option, the virtual size of the frame might exceed the physical space allocated for it in the window. If that happens, scrolling is enabled for the frame. If you omit this option, the physical and virtual size of the frame are always the same and scrolling is never enabled for the frame.

SIDE-LABELS

Displays field labels to the left of and centered against the data, separated from the data by a colon (:) and a space. If you do not use the SIDE-LABELS option, Progress displays labels above their corresponding fields in the frame header and separates the labels from the field values with underlining.

size-phrase

Specifies the size of the frame. This is the syntax for *size-phrase*:

Syntax

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information on *size-phrase*, see the SIZE phrase reference entry.

STREAM *stream*

Allows you to specify the name of a stream for SQL statements.

THREE-D

Specifies that the frame and all contained widget appear in three-dimensional format (Windows only). If you specify the THREE-D option for a frame, the default background color is gray rather than the window color. Frames do not inherit the THREE-D setting from a parent window, and child frames do not inherit the THREE-D setting from a parent frame.

title-phrase

Displays a title as part of the top line of the box around a display frame. Following is the syntax for the *title-phrase*:

Syntax

```
TITLE
  [   { [ BGCOLOR expression ]
        [ DCOLOR expression ]
        [ FGCOLOR expression ]
      }
    | COLOR color-phrase ]
  [ FONT expression ] title-string
```

The *title-string* is a constant, field name, variable name, or expression whose result is a character value. The *expression* is the value you want to display as a title. If *title-string* is a constant character string, it must be surrounded by quotes (""). Progress automatically centers *title-string* in the top line of the frame box.

You can use the BGCOLOR and FGCOLOR options to specify the background and foreground color of the title in a graphical interface. You can use the DCOLOR option to specify the color of the title in a character interface. The COLOR option is obsolete and is retained only for backward compatibility.

TOP-ONLY

Indicates that no other frame can overlay this frame. If you do not use this option, other frames that use the OVERLAY option can overlay this frame. If Progress has to display an OVERLAY frame and by doing so will partially obscure a TOP-ONLY frame, it first hides the TOP-ONLY frame. For more information, see the OVERLAY statement reference entry.

USE-TEXT

Specifies that the default widget type for all widgets in the frame is TEXT rather than FILL-IN. Thus, all border padding on the widgets is dropped.

```
V6FRAME [ USE-REVVIDEO | USE-UNDERLINE]
```

The V6FRAME option is designed specifically to compile and run Progress Version 6 applications with Progress Version 7 or later in Windows. This option uses the V6FontNumber setting in the [Startup] section of the current environment to calculate the height and width of a character unit and then set the layout grid used to compile frames for display in Progress Version 7 or later.

At run time, the FONT attribute for a frame compiled with the V6FRAME option is set to the font number specified with the V6FontNumber setting. The default setting for the V6FontNumber setting is 3.

By default, V6FRAME displays a border around a fill-in field. This means that your code requires more space on the screen than in Progress Version 6. You can override this behavior with one of the following options.

- USE-REVVIDEO displays no border around a fill-in field. When a fill-in is enabled for input, the color of the fill-in changes to the color specified with the INPUT setting in the [Colors] section in the current environment. The IBEAM cursor signals that a fill-in field has input focus.
- USE-UNDERLINE displays no border around a fill-in widget. When a fill-in is enabled for input, the underline attribute of the font (V6FontNumber) for the fill-in is turned on. The color of a fill-in enabled for input does not change. The IBEAM cursor signals that a fill-in field has input focus.

The V6FRAME option also limits the vertical size of a frame title to one character unit based upon the layout grid. The text of the frame title is in the font specified with the V6FontNumber setting in the [Startup] section of the current environment.

The V6FRAME option governs the appearance of screen output only. Use the STREAM-IO option to compile procedures that output to files and printers. If you specify the V6FRAME and STREAM-IO options in the same frame phrase, the STREAM-IO option overrides the V6FRAME option.

For more information on the environment for an OpenEdge session, see *OpenEdge Deployment: Managing 4GL Applications*.

VIEW-AS DIALOG-BOX

Specifies that the frame is displayed as a dialog box. A dialog box is a modal, one-down frame with many of the properties of a window. Like a window, a dialog box can be moved and programmatically resized, and it acquires scroll bars when it is resized smaller than its original frame dimensions. Unlike a window, it cannot be minimized or maximized; nor can it have a menu bar. As a frame-level widget, it is owned by a window and can contain a frame family, but it cannot be owned by another frame or dialog box. Because it is modal, a dialog box must be disabled before any other widgets in the application can be accessed by the user. For more information on the properties of a dialog box, and to compare them with the properties of a frame, see the "Widget Reference" section on page 1311.

WIDTH *n*

Specifies the number (*n*) of columns in a frame. If you do not use *size-phrase* or the WIDTH option, the width of the frame is based on the fields you are displaying, the position of the frame, and the width of the current or specified window.

IN WINDOW *window*

Specifies the window in which the frame is displayed. The value *window* must be the handle of a window. This option is not allowed in a DISABLE statement. By default, Progress displays the frame in the current window.

Examples

The r-frame.p procedure displays the cust-num, name, and phone number for each customer record. The frame phrase (starting with the word WITH) describes the frame being used to display that information.

r-frame.p

```
FOR EACH customer:
  FORM HEADER
    "No-box, No-Underline, No-labels, 5 DOWN"
    SKIP   "Centered" SKIP(2)
    WITH NO-BOX NO-UNDERLINE NO-LABELS
      CENTERED 5 DOWN.
  DISPLAY cust-num name phone.
END.
```

The r-frame2.p procedure produces a customer report, using Customer List as the header for each page of the report and using Customer List Continued On Next Page as the footer for each page of the report. The OUTPUT TO statement directs all output to the file phone.lst. After running the r-frame2.p procedure, you can press **GET** then type the name of the file to view the contents of phone.lst.

r-frame2.p

```
OUTPUT TO phone.lst PAGED PAGE-SIZE 20.
FOR EACH customer:

  FORM HEADER "Customer List" AT 1 PAGE-NUMBER
    TO 60 WITH FRAME hdr PAGE-TOP
     CENTERED NO-BOX.
  VIEW FRAME hdr.
  FORM "Customer List Continued On Next Page"
    WITH FRAME footr PAGE-BOTTOM CENTERED.
  VIEW FRAME footr.

  DISPLAY cust-num name phone WITH CENTERED.
END.
HIDE FRAME footr.
OUTPUT CLOSE.
```

Notes

- PAGE-TOP and PAGE-BOTTOM frames are activated based on DISPLAY or VIEW statements as previously described. They are deactivated when the block in which the frames are scoped iterates or ends.
- If you use the SIZE phrase for a down frame, then the size you specify determines the number of iterations in the frame. The number of iterations you specify with the DOWN option is ignored.
- You can input and output to a frame only when that frame is in full view. Therefore, when you input or output to a frame that is hidden or partially overlayed, Progress displays the frame first.
- An empty WITH clause is valid. If the WITH keyword appears by itself, or in the clause following an earlier WITH, it is ignored. This feature is useful when designing template programs to be called with arguments. For example, a template program with a line like DISPLAY {1} WITH {2} executes correctly even if called with only one argument.
- The SIZE phrase and WIDTH options are mutually exclusive. If you specify WIDTH or you specify neither WIDTH nor the SIZE, the height of a frame is based on the fields you are displaying, the position of the frame, and whether or not it is a down frame.
- A frame parented by another frame cannot function as a down frame.

- If you position a child frame completely outside the virtual area of its parent frame, Progress raises ERROR at run time when the frame is realized.
- If you position a child frame partially within the virtual area of its parent frame or the child frame is larger than the virtual area of the parent frame, Progress crops the child frame to fit the parent's virtual area and adds scroll bars to the child.
- If you position a child frame partially within the physical area of its parent frame or the child frame is larger than the physical area of the parent frame, Progress adds scroll bars to the parent.
- You cannot specify the VIEW-AS DIALOG-BOX option for a frame used as a DDE frame. For information on DDE frames, see *OpenEdge Development: Programming Interfaces*.
- If you have enabled application-defined widget IDs in your OpenEdge GUI application, by specifying the Use Widget ID (`-usewidgetid`) startup parameter, then Progress uses the value specified in the WIDGET-ID option to set the WIDGET-ID attribute for this widget when it creates the widget at runtime, instead of using the widget ID it normally generates by default. If you have not enabled application-defined widget IDs, then Progress ignores this option setting at runtime.

  For more information about the WIDGET-ID attribute, see its reference entry in the "Attributes and Methods Reference" section on page 1497. For more information about the Use Widget ID (`-usewidgetid`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- See *OpenEdge Development: Progress 4GL Handbook* for more information on frames.
- For SpeedScript, WebSpeed evaluates the Frame phrase as though you were running a character client. The typical WebSpeed application does not use frames when defining layout. However, if you are using existing Progress code that includes frame layouts, you can iterate through frame children to retrieve validation expressions and help strings. Generally, in SpeedScript programming, the frame serves as a virtual container for widgets. These options are invalid: ATTR-SPACE, NO-ATTR-SPACE, CENTERED, CONTEXT-HELP, CONTEXT-HELP-FILE, DEFAULT-BUTTON, SCROLLBAR VERTICAL, V6FRAME, USE-REVVIDEO, USE-UNDERLINE, VIEW-AS DIALOG-BOX, IN WINDOW.

See also DEFINE FRAME statement, FORM statement, Format phrase, FRAME-COL function, FRAME-DOWN function, FRAME-LINE function, FRAME-ROW function

FRAME-COL function

Returns a decimal value that is the column position of the left corner of a frame within its window.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-COL [ ( frame ) ]
```

frame

The name of the frame whose column position you are trying to determine. If you do not supply a frame name, the FRAME-COL function uses the default frame for the block it is in. If the FRAME-COL function is in a DO block, the function uses the default frame scoped to the block containing the DO block.

Example

This procedure displays customer information in one frame, then displays order information in an overlay frame. FRAME-ROW places the overlay frame on the ninth row of the second column. FRAME-COL places the overlay frame on the first column of the first frame.

r-frcol.p

```
FOR EACH customer:
   DISPLAY customer WITH FRAME cust-frame 2 COLUMNS
     TITLE "CUSTOMER INFORMATION".
   FOR EACH order OF customer:
       DISPLAY order-num order-date ship-date promise-date carrier
               instructions po
         WITH 2 COLUMNS 1 DOWN OVERLAY TITLE "CUSTOMER'S ORDERS"
         ROW FRAME-ROW(cust-frame) + 8
         COLUMN FRAME-COL(cust-frame) + 1.
   END.
END.
```

Notes

- The FRAME-COL function returns a value of 0 if the frame you specify is not in view when Progress evaluates the function.
- To convert the decimal value returned by FRAME-COL to an integer value, use the INTEGER function.

See also

Frame phrase, FRAME-DOWN function, FRAME-LINE function, FRAME-ROW function, INTEGER function

FRAME-DB function

Returns the logical database name for the field where the cursor is.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-DB
```

The function requires no arguments. If the cursor is in a field that is not a database field, this function displays no value for the field.

Example

For each field being updated, this procedure displays the field name, the table the field belongs to, and the database in which the table exists. The EDITING phrase is part of the UPDATE statement; it displays information on the field as you update the record, and then reads each of the keystrokes entered (READKEY) and applies those keystrokes (APPLY LASTKEY).

r-frdb.p

```
FOR EACH customer:
UPDATE cust-num name address address2 city
    state postal-code WITH 1 DOWN 1 COLUMN CENTERED
     EDITING:
        DISPLAY
          "You are editing field: " FRAME-FIELD SKIP
         " of file: " FRAME-FILE SKIP
         " In database: " FRAME-DB
         WITH FRAME a ROW 15 NO-LABELS CENTERED.
        READKEY.
        APPLY LASTKEY.
      END.  /*Editing*/
END.
```

Notes

- If the cursor is not in an enabled input field when the last input statement is executed, or the input field is not associated with a database field, FRAME-DB returns an empty string.
- Use this syntax to find the name of a schema holder for a non-OpenEdge database:

```
SDBNAME ( FRAME-DB )
```

See also

DBCODEPAGE function, DBCOLLATION function, FRAME-FIELD function, FRAME-FILE function, FRAME-INDEX function, LDBNAME function, PROGRAM-NAME function

FRAME-DOWN function

Returns an integer value that represents the number of iterations in a frame.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-DOWN [ ( frame ) ]
```

frame

The name of the frame whose number down you are trying to determine. If you do not supply a frame name, the FRAME-DOWN function uses the default frame for the block it is in. If the FRAME-DOWN function is in a DO block, the function uses the default frame scoped to the block containing the DO block.

Example

This procedure displays customers in a frame. When the frame is full, the procedure prompts "Do you want to see the next page?" The procedure recognizes that the frame is full when the value of FRAME-LINE (current logical line number) equals the value of FRAME-DOWN (number of iterations in the frame).

r-frdown.p

```
DEFINE VARIABLE ans AS LOGICAL.

REPEAT:
  FIND NEXT customer.
  DISPLAY cust-num name.
  IF FRAME-LINE = FRAME-DOWN
  THEN DO:
    MESSAGE "Do you want to see the next page ?"
    UPDATE ans.
    IF NOT ans
    THEN LEAVE.
  END.
END.
```

Note

The FRAME-DOWN function returns a value of 0 if used with a single frame or if the frame is not in view when the function is evaluated.

See also

Frame phrase, FRAME-COL function, FRAME-LINE function, FRAME-ROW function

FRAME-FIELD function

During a data entry statement, returns the name of the input field the cursor is in. At other times, returns the name of the input field the cursor was last in.

The FRAME-FIELD function is particularly useful if you want to provide the user with help for the input field being used.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-FIELD
```

Example

For each field the user is updating, this procedure displays the name of the field, the table the field belongs to, and the value currently in the field. The EDITING phrase is part of the UPDATE statement; it displays information on the field as the user updates the record, and then reads each of the keystrokes entered (READKEY) and applies those keystrokes (APPLY LASTKEY).

r-frfld.p

```
FOR EACH customer:
    UPDATE cust-num name address address2 city state postal-code
         WITH 1 DOWN 1 COLUMN CENTERED
   EDITING:
      DISPLAY "You are editing field:" FRAME-FIELD SKIP
              "Of file:" FRAME-FILE SKIP
              "Its value is:" FRAME-VALUE FORMAT "x(20)"
              WITH FRAME a ROW 15 NO-LABELS CENTERED.
      READKEY.
      APPLY LASTKEY.
   END. /* Editing */
END.
```

Notes

- If the current or last input field is an array, FRAME-FIELD returns the name of the field but does not indicate the array element that the input field represents. To display the array element, use the FRAME-INDEX function.
- If the cursor was not in an enabled input field when the last input statement ended, FRAME-FIELD returns an empty string.

See also

FRAME-FILE function, FRAME-INDEX function, FRAME-VALUE function, PROGRAM-NAME function

FRAME-FILE function

Returns the name of the database table that contains the field the cursor is in. The FRAME-FILE function is useful if you want to provide users with context-sensitive help.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-FILE
```

Example

This procedure updates fields from the order table and the customer table. It uses the FRAME-FILE function to tell you which table contains the field being updated.

r-frfile.p

```
FOR EACH customer, EACH order OF customer:
      DISPLAY order-num WITH CENTERED ROW 2 FRAME onum.
      UPDATE
        customer.cust-num AT 5 order.cust-num AT 30 SKIP
        customer.name  AT 5
        customer.city AT 5
        customer.state AT 5
        customer.postal-code AT 5
        WITH ROW 8 CENTERED 1 DOWN NO-LABELS
        EDITING:
         MESSAGE "     The field" FRAME-FIELD "is from the" FRAME-FILE
                 "file     ".
         READKEY.
         APPLY LASTKEY.
      END.  /* Editing */
END.
```

Notes

- FRAME-FILE returns a null string if the frame field being entered is not associated with a database field.
- If the cursor is not in an enabled input field when the last input statement ends, FRAME-FILE returns a null string.
- The FRAME-FILE value is set to blanks at the next PAUSE statement, at the next READKEY statement, or when Progress pauses automatically.

See also

FRAME-FIELD function, FRAME-VALUE function, PROGRAM-NAME function

FRAME-INDEX function

During a data entry statement, returns the subscript of the array element of the input field that the cursor is in. At other times, returns the subscript of the array element the cursor was in.

The FRAME-INDEX function is particularly useful if you want to provide the user with help for the input array element being edited.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-INDEX
```

Example

In this example, the FRAME-INDEX function uses the cursor position to determine which option you have chosen:

r-frindx.p

```
DEFINE VARIABLE menu AS CHARACTER EXTENT 3.
DO WHILE TRUE:
  DISPLAY
    "1. Display Customer Data" @ menu[1] SKIP
    "2. Display Order Data"    @ menu[2] SKIP
    "3. Exit"                  @ menu[3] SKIP
    WITH FRAME choices NO-LABELS.
  CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
    TITLE "Demonstration Menu" WITH CENTERED ROW 10.
  HIDE FRAME choices.
  IF FRAME-INDEX EQ 1 THEN
    MESSAGE "You picked option 1.".
  ELSE IF FRAME-INDEX EQ 2 THEN
    MESSAGE "You picked option 2.".
  ELSE IF FRAME-INDEX EQ 3 THEN LEAVE.
END.
```

Notes

- If the cursor is not in an enabled input field when the last input statement is executed, FRAME-INDEX returns a 0. For example, FRAME-INDEX returns 0 if the user presses **END-ERROR** on the previous input statement.
- The FRAME-INDEX value is set to 0 at the next pause (done by a PAUSE statement or automatically by Progress) or at the next READKEY statement.
- If you are updating a subset of an array-for example, `TEXT(array-field[i TO 12])`, and you use a variable subscript `(i)`, then FRAME-INDEX returns 0. If you use a constant subscript, `TEXT(array-field[6 TO 12])`, then FRAME-INDEX returns the correct value.

See also Frame phrase, FRAME-DB function, FRAME-FIELD function, FRAME-FILE function

FRAME-LINE function

Returns an integer value that represents the current logical line number in a down frame.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-LINE [ ( frame ) ]
```

frame

The frame name that you are trying to determine a line number for. If you do not supply a frame name, the FRAME-LINE function uses the default frame for the block that contains the FRAME-LINE function. If the FRAME-LINE function is in a DO block, the function uses the default frame scoped to the block that contains the DO block.

Example

This procedure lists customers and allows the user to delete customers one at a time. When the user presses **GET** to delete a customer, the procedure displays an overlay frame below the last customer displayed. The overlay frame prompts "Do you want to delete this customer?" The user answers yes or no. Progress calculates the position of the overlay frame from the upper-right corner of the frame and the current line within the frame. That is, FRAME-ROW + 3 + FRAME-LINE gives the position of the current line in the frame, taking into account the three lines for the frame box and the labels. The prompt is placed five lines below the current line.

r-frline.p

```
DEFINE VARIABLE ans AS LOGICAL
                    LABEL "Do you want to delete this customer ?".

IF KBLABEL("GET") = "GET"
THEN ON F3 GET.

STATUS INPUT "Enter data, or use the " + KBLABEL("get")
           + " key to delete the customer".

get-cust:
      FOR EACH customer WITH 10 DOWN:
        UPDATE cust-num name credit-limit
       editing:
          READKEY.
          IF KEYFUNCTION(lastkey) = "get"
          THEN DO:
           UPDATE ans WITH ROW FRAME-ROW + 3 + FRAME-LINE + 5
                    COLUMN 10 SIDE-LABELS OVERLAY FRAME del-frame.
           IF ans
           THEN DO:
              DELETE customer.
              NEXT get-cust.
          END.
        END.
        APPLY LASTKEY.
     END.
END.
```

Notes

- If there is a down pending for a frame, the FRAME-LINE function returns a value equal to FRAME-LINE + 1.
- The FRAME-LINE function counts an underline row as a logical line. A logical line corresponds to one iteration in a down frame and can contain more than one physical line.
- The FRAME-LINE function returns a value of 0 if the frame is not in view when the function is evaluated.

See also Frame phrase, FRAME-COL function, FRAME-DOWN function, FRAME-ROW function

FRAME-NAME function

Returns the name of the frame that the cursor is in to a field that is enabled for input.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-NAME
```

Example

This procedure displays customer information in one frame, then displays order information for the customer in a second frame. Use the FRAME-NAME function to display the name of the frame the cursor is in.

r-frname.p

```
FOR EACH customer, EACH order OF customer:
    DISPLAY order-num WITH CENTERED ROW 2 FRAME onum.
    UPDATE customer.cust-num AT 5 customer.name AT 30 SKIP
        WITH FRAME custfrm WITH CENTERED 1 DOWN
        EDITING:
            DISPLAY " You are currently editing a frame called "
                    FRAME-NAME WITH FRAME d1 WITH 1 DOWN CENTERED.
            READKEY.
            APPLY LASTKEY.
            IF LASTKEY = KEYCODE("RETURN") THEN
                 MESSAGE " Press the space bar to edit order shipdate".
         END. /* Editing */
    HIDE FRAME custfrm.
    HIDE FRAME d1.
    UPDATE ship-date AT 5 WITH FRAME orderfrm WITH CENTERED 1 DOWN
       EDITING:
         DISPLAY " Now you are editing a frame called"
                 FRAME-NAME WITH FRAME d2 WITH 1 DOWN CENTERED.
         READKEY.
         APPLY LASTKEY.
       END.
    HIDE FRAME orderfrm.
    HIDE FRAME d2.
END.
```

Notes

- The FRAME-NAME function returns an empty string for a frame that has not been named (the default frame). It also returns an empty string if the cursor is in a field that is not enabled for input.
- When using the FRAME-NAME function, you must place it logically following the Frame phrase where it is named.
- FRAME-NAME is especially useful for context-sensitive help.

See also Frame phrase, PROGRAM-NAME function

FRAME-ROW function

Returns a decimal value that represents the row position of the upper-left corner of a frame within its window.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-ROW [ ( frame ) ]
```

frame

The name of the frame whose row position you are trying to determine. If you do not supply a frame name, the FRAME-ROW function uses the default frame for the block that contains the FRAME-ROW function. If the FRAME-ROW function is in a DO block, the function uses the default frame scoped to the block that contains the DO block.

Example

This procedure displays customer information in one frame, then displays order information for the customer in a second frame that overlays the first. FRAME-ROW and FRAME-COL control the placement of the overlay frame. FRAME-ROW places the overlay frame on the eighth row of the first frame. FRAME-COL places the overlay frame on the first column of the first frame.

r-frrow.p

```
FOR EACH customer:
   DISPLAY customer WITH FRAME cust-frame 2 COLUMNS
        TITLE "CUSTOMER INFORMATION".
   FOR EACH order OF customer:
       DISPLAY order-num order-date ship-date promise-date carrier
            instruction po
            WITH 2 COLUMNS 1 DOWN OVERLAY TITLE "CUSTOMER'S ORDERS"
            ROW FRAME-ROW(cust-frame) + 8
            COLUMN FRAME-COL(cust-frame) + 1.
   END.
END.
```

Note

To convert the decimal value returned by FRAME-ROW to an integer value, use the INTEGER function.

See also

Frame phrase, FRAME-COL function, FRAME-DOWN function, FRAME-LINE function, INTEGER function

FRAME-VALUE function

During a data entry statement, returns the (character string) value of the input field that the cursor is in to the current input field. At other times, returns the (character string) value of the input field the cursor was last in.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-VALUE
```

Example

When the user presses **END-ERROR** while running this procedure, the procedure displays the name and value of the field the user was updating, along with the name of the table that contains that field.

r-frval.p

```
FOR EACH customer:
  DISPLAY cust-num name address city state postal-code
    WITH 1 COLUMN DOWN COLUMN 20.
  SET address city state postal-code.
END.

DISPLAY
  "You were updating field:" FRAME-FIELD FORMAT "x(20)" SKIP
  "Of file:" FRAME-FILE SKIP
  "And it had the value:" FRAME-VALUE FORMAT "x(20)" SKIP(2)
  "When you pressed the END-ERROR key"
  WITH FRAME helper NO-LABELS COLUMN 20 ROW 14 NO-BOX.
```

Notes

- If the cursor is not in an enabled input field when the last input statement ends, FRAME-VALUE returns a null string.
- FRAME-VALUE is set to blanks at the next pause (done by a PAUSE statement or automatically by Progress) or at the next READKEY statement.
- FRAME-VALUE returns strings. If you use FRAME-VALUE to return a number, you must convert it prior to numeric comparisons. For example:

```
FIND customer WHERE cust-num=INTEGER(FRAME-VALUE)
```

See also

FRAME-FIELD function, FRAME-FILE function, FRAME-VALUE function, PROGRAM-NAME function

FRAME-VALUE statement

Stores the value of an expression in a frame field during a data entry statement.

Note: Does not apply to SpeedScript programming.

Syntax

```
FRAME-VALUE = expression
```

expression

A constant, field name, variable name or expression whose value you want to store in a frame field. If no frame is active when Progress runs this statement, Progress returns an error message. Otherwise, if the frame is in view, Progress redisplays the field.

The data type of the *expression* must be the same as the data type of the frame field in which you are storing that expression. However, if the data type of *expression* is character, Progress stores characters in the frame field regardless of the data type of that frame field, truncating characters if necessary.

The FRAME-VALUE statement can pass information from an `applhelp.p` procedure to the calling procedure. For example, if the user enters a value into a field called help-field, you can pass that value back to the calling procedure with this statement:

```
FRAME-VALUE = INPUT help-field.
```

Example This procedure displays the word Progress, the date, and a message instructing you to enter data or press the **GET** key to enter the Unknown value (?). You can update the information in the frame. If you press GET, the r-frmval.p procedure assigns the Unknown value (?) to a field with the FRAME-VALUE statement.

r-frmval.p

```
DEFINE VARIABLE txt AS CHARACTER INITIAL "PROGRESS".
DEFINE VARIABLE tmpdate AS DATE INITIAL TODAY.IF KBLABEL("GET") = "GET"
THEN ON F3 GET.STATUS INPUT "Enter data or use the "
           + KBLABEL("GET")
           + " key to enter the unknown value (?)".
UPDATE txt tmpdate
  EDITING:
     READKEY.
     IF KEYFUNCTION(LASTKEY) = "GET"
     THEN DO:
        FRAME-VALUE = ?.
        NEXT.
     END.
     APPLY LASTKEY.
  END.
```

Note For more information on frames, see *OpenEdge Development: Progress 4GL Handbook*.

See also FRAME-FIELD function, FRAME-FILE function, FRAME-VALUE function

FUNCTION statement

Defines or forward declares a user-defined function.

You can also use the FUNCTION statement to invoke a Web service operation. For more information on invoking Web service operations, see *OpenEdge Development: Web Services*.

Syntax

```
FUNCTION function-name [ RETURNS ] data-type [ PRIVATE ]
  [ ( parameter [, parameter ] ... ) ]
  {     FORWARD
      | [ MAP [ TO ] actual-name ] IN proc-handle
      | IN SUPER
  }
```

Use the following syntax to invoke a Web service operation:

```
FUNCTION operationName [ RETURNS ] data-type
  [ ( parameter [ , parameter ] ... ) ]
  IN hPortType .
```

function-name

The name of the function. You must avoid Progress reserved keywords. For a list of Progress keywords, see the Keyword Index in this manual.

[RETURNS] *data-type*

Indicates the function returns a value, and specifies the data type of that return value. Progress provides the following return value data types: CHARACTER, CLASS, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INTEGER, LOGICAL, LONGCHAR, MEMPTR, RAW, RECID, ROWID, and WIDGET-HANDLE.

You specify a class or interface as a return value for a user-defined function using the following syntax:

```
[ CLASS ] { type-name }
```

Progress passes the object reference associated with the class or interface (by value), not the class or interface itself.

type-name

A character string that specifies the type name of the class or interface. Specify a type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

If the specified class or interface type name conflicts with an abbreviation of a built-in Progress data type name, such as INT for INTEGER, you must specify the CLASS keyword.

Note: If you invoke a function on an AppServer, the function cannot return a value as a LONGCHAR, MEMPTR, or CLASS.

PRIVATE

Indicates the following about the user-defined function:

- That it cannot be invoked from an external procedure-that is, from a procedure file external to the current procedure file.
- That the INTERNAL-ENTRIES attribute on the procedure that defines it does not provide its name (unless the procedure that defines it is the current procedure file).
- That the GET-SIGNATURE method on the procedure that defines it does not provide its signature (unless the procedure that defines it is the current procedure file).

```
( parameter [, parameter ] ... )
```

Defines one or more parameters of the function.

For the parameter definition syntax, see the Parameter definition syntax reference entry in this book.

FORWARD

Lets a procedure reference a user-defined function whose definition has not yet appeared. You must use this option when the definition of the function appears later in the procedure.

The FUNCTION statement with the FORWARD option must include the following information on the function: the data type it returns, and the data type and mode (INPUT, OUTPUT, or INPUT-OUTPUT) of each parameter.

This entry uses the term **forward declaration** to refer to statements such as the FUNCTION statement with the FORWARD option. This entry also mentions **forward declaring** user-defined functions.

If you forward declare a user-defined function, reference it, and do not define it before the end of the procedure, the compiler reports an error.

IN *proc-handle*

Indicates that the function's definition resides in another procedure. The *proc-handle* parameter represents an expression that evaluates to a handle to the procedure that defines the function. This procedure can be an active procedure in the local context or a remote persistent procedure. For more information on remote user-defined functions, see *OpenEdge Application Server: Developing AppServer Applications*.

The FUNCTION statement with the IN option must include the following information on the function: the data type it returns, and the data type and mode (INPUT, OUTPUT, or INPUT-OUTPUT) of each parameter.

[MAP **[** TO **]** *actual-name* **]** IN *proc-handle*

Indicates three things: that *function-name* (the second element in the FUNCTION statement) is an alias (alternative name) of the function, that *actual-name* is the name that appears in the definition of the function, and that the definition of the function resides in another procedure. *actual-name* represents the actual name of the function. *proc-handle* represents an expression that evaluates to a handle to the procedure that defines the function.

Note: The MAP option might simplify your code if it references two different user-defined functions that have the same name but that reside in different procedures.

`IN SUPER`

Indicates that the implementation of the function resides in a super procedure. For more information on super procedures, see *OpenEdge Development: Progress 4GL Handbook*.

`operationName`

The name of a Web service operation specified in a WSDL file.

`hPortType`

A handle to a procedure object that encapsulates a Web service operation.

Examples

The first example, `r-udf1.p`, defines and references the user-defined function doubler(), which accepts an integer and returns the integer multiplied by two.

r-udf1.p

```
/* r-udf1.p */
/* Defines and references a user-defined function */

/* Define doubler() */
FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER).
    RETURN (2 * parm1).
    END FUNCTION.

/* Reference doubler() */
DISPLAY  "doubler(0)=" doubler(0) skip
  "doubler(1)=" doubler(1) skip
  "doubler(2)=" doubler(2) skip.
```

The second example, r-udf2.p, forward declares, references, and defines doubler().

r-udf2.p

```
/* r-udf2.p */
/* Forward-declares, references, and defines a user-defined function */

/* Forward declare doubler() */
FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER) FORWARD.

/* Reference doubler() */
DISPLAY "doubler(0)=" doubler(0).
DISPLAY "doubler(1)=" doubler(1).
DISPLAY "doubler(2)=" doubler(2).

/* Define doubler() */
FUNCTION doubler RETURNS INTEGER.
    RETURN (2 * parm1).
    END FUNCTION.
```

The third example consists of two procedures, r-udf3.p and r-udfdef.p. The example illustrates defining a user-defined function in an external procedure.

The first procedure, r-udf3.p, runs the first procedure persistently, declares doubler(), references it, and deletes the persistent procedure.

r-udf3.p

```
/* r-udf3.p */
/* references an externally-defined user-defined function */

/* define items */
DEFINE VARIABLE myhand AS HANDLE.
DEFINE VARIABLE mystr  AS CHARACTER FORMAT "x(20)".

/* forward declare doubler() */
FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER) IN myhand.

/* run the procedure that defines doubler() */
RUN src\prodoc\langref\r-udfdef.p PERSISTENT SET myhand.

/* reference doubler() */
DISPLAY  "doubler(0)=" doubler(0) skip
        "doubler(1)=" doubler(1) skip
        "doubler(17)=" doubler(17) skip.

/* delete the procedure that defines doubler */
DELETE PROCEDURE(myhand).
```

The second procedure, r-udfdef.p, defines doubler().

r-udfdef.p

```
/* r-udfdef.p */
/* Defines user-defined function doubler() */

FUNCTION doubler RETURNS INTEGER (INPUT parm1 AS INTEGER).
  RETURN (2 * parm1).
  END FUNCTION.
```

To start the third example, run r-udf3.p in the Procedure Editor.

In the fourth example, r-fctrl2.p, the user-defined function fact() implements the factorial function, common in probability and statistics, and commonly notated "!" (6! = 6 x 5 x 4 x 3 x 2; 100! = 100 x 99 x 98 x ... x 2).

r-fctrl2.p

```
/* r-fctrl2.p */
/* demonstrates user-defined function fact() */

FUNCTION fact RETURNS INTEGER (val AS INTEGER):
  IF val LT 0 THEN RETURN 0.
  IF val LE 1 THEN RETURN 1.
  RETURN val * fact(val - 1).
END.

DEFINE VARIABLE inp AS INTEGER LABEL "Input Value".
REPEAT:
  UPDATE inp WITH TITLE "Factorials".
  DISPLAY fact(inp) LABEL "Factorial".
END.
```

Notes

- Before you reference a user-defined function, you must forward declare it, declare it external (by using FUNCTION statement's IN option), or define it.
- To declare a function forward or external, you must specify at minimum the function name and return type, and for each parameter, the mode and data type.
- To define a function you have forward declared, you must specify at minimum the function name, the return type, the parameters (if any), and the logic.

- A period must appear after the FUNCTION statement and after each logic statement.
- A user-defined function's logic must not reference, either directly or indirectly, statements that block I/O (namely, the CHOOSE, INSERT, PROMPT-FOR, READKEY, SET, UPDATE, and WAIT-FOR statements).
- You cannot define shared objects, work tables, temporary tables, or ProDataSet objects within a user-defined function.
- Progress implements scalar and array parameters of user-defined functions as NO-UNDO variables.
- A reference to a user-defined function must match the forward declaration or definition with respect to the return type, and with respect to the number, type, and mode of the parameters.
- When a Progress 4GL predicate (such as a WHERE clause) contains a user-defined function, Progress evaluates the function once-when it opens the query or enters the FOR EACH block.
- When Progress encounters a user-defined function declared external that references a user-defined function declared external that references a user-defined function declared external, etc., Progress tolerates up to 64 levels of indirection. At the 65th level, Progress issues an error message and returns the Unknown value (?).
- If a user-defined function has one or more buffer parameters and its definition resides in another procedure, the referencing procedure and the defining procedure must reside on the same machine. If a user-defined function does not have buffer parameters, the invoking procedure and the defining procedure can reside on different machines.
- When you invoke a user-defined function (or a built-in function), you need not assign the function's return value to a variable. You might use this technique with a function that performs some action on a persistent object, such as a shared variable, when you want the action to occur and do not want to check the return value. For example:

```
doubler(my-shared-variable).
```

- When you invoke a user-defined function, you may pass a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter by value, by reference, or by binding using the BY-VALUE, BY-REFERENCE, or BIND keyword, respectively. For example:

```
myfunc(OUTPUT TABLE tt BY-REFERENCE).
```

  For more information about passing these parameters by value, by reference, or by binding, see the Parameter passing syntax reference entry in this book.

See also COMPILE statement, DYNAMIC-FUNCTION function, GET-SIGNATURE( ) method, INTERNAL-ENTRIES attribute, Parameter definition syntax, PROGRAM-NAME function, RETURN statement

GATEWAYS function

The GATEWAYS function has been replaced by the DATASERVERS function, which is exactly equivalent.

This function is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
GATEWAYS
```

GE or >= operator

Returns a TRUE value if the first of two expressions is greater than or equal to the second expression.

Syntax

```
expression { GE | >= } expression
```

expression

A constant, field name, variable name, or any combination of these. The expressions on either side of the GE or >= must be of the same data type, although one might be integer and the other decimal.

Example

This procedure displays item information for those items whose on-hand value is greater than or equal to 120:

r-ge.p

```
FOR EACH item WHERE on-hand >= 120:
  DISPLAY item.item-num item-name on-hand.
END.
```

Notes

- By default, Progress uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).
- If either of the expressions is the Unknown value (`?`), then the result is the Unknown value (`?`); if both expressions are the Unknown value (`?`), then the result is TRUE.

- You can compare character strings with GE. Most character comparisons are case insensitive in Progress. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *expression* is a field or variable defined as case sensitive, the comparison is case sensitive and “Smith” does not equal “smith”.

- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ’ , but lowercase a is greater than these.

- You can use GE to compare DATE, DATETIME, and DATETIME-TZ data. The data type that contains less information (that is, a DATE value contains less information than a DATETIME value, and a DATETIME value contains less information than a DATETIME-TZ value) is converted to the data type with more information by defaulting the time value to midnight, and the time zone value to the session's time zone (when the data type does not contain the time or time zone). Comparisons with DATETIME-TZ data are based on Coordinated Universal Time (UTC) date and time.

- You can use GE to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or Progress raises a run-time error.

- You cannot use GE to compare one CLOB field to another.

GENERATE-PBE-KEY function

Generates a password-based encryption key, based on the PKCS#5/RFC 2898 standard, and returns the key as a RAW value.

Syntax

```
GENERATE-PBE-KEY( password [ , salt ] )
```

password

The password (a binary value) to use in generating the encryption key. This value may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the password contains a CHARACTER or LONGCHAR value, Progress converts it to UTF-8 (which ensures a consistent value regardless of code page settings) before using it to generate the encryption key. To avoid this automatic conversion, specify a RAW or MEMPTR value. If you specify the Unknown value (?), the result is the Unknown value (?).

salt

An optional RAW expression that evaluates to the salt value (a random series of 8 bytes) to use in generating the encryption key. If you specify the Unknown value (?), the current value of the ENCRYPTION-SALT attribute is used. If no salt value is specified in the ENCRYPTION-SALT attribute, no salt value is used.

You can also use the GENERATE-PBE-SALT function to generate a salt value, which can help to ensure that the password key value is unique.

If specified, this salt value is combined with the password value and hashed some number of times to generate a password-based encryption key (using the algorithm specified by the PBE-HASH-ALGORITHM attribute and the number of iterations specified by the PBE-KEY-ROUNDS attribute).

Notes

- You are responsible for generating, storing, and transporting these values.
- The size of the generated encryption key is determined by the cryptographic algorithm specified by the SYMMETRIC-ENCRYPTION-ALGORITHM attribute.
- Before invoking this function, be sure to set the PBE-HASH-ALGORITHM attribute to the name of the hash algorithm to use.
- If you call this function multiple times with the same password string, hash algorithm, number of iterations, and salt value, the same binary key is generated each time.

See also

GENERATE-PBE-SALT function, GENERATE-RANDOM-KEY function, MD5-DIGEST function, SHA1-DIGEST function, SECURITY-POLICY system handle

GENERATE-PBE-SALT function

Generates a random salt value (a series of 8 bytes) to use in generating an encryption key, and returns the salt value as a RAW value. Using a salt value can help to ensure that a password key value is unique.

Syntax

```
GENERATE-PBE-SALT
```

Notes

- This salt value is combined with a password value and hashed some number of times to generate a password-based encryption key (using the algorithm specified by the PBE-HASH-ALGORITHM attribute and the number of iterations specified by the PBE-KEY-ROUNDS attribute).
- You are responsible for generating, storing, and transporting this value.

See also

GENERATE-PBE-KEY function, MD5-DIGEST function, SHA1-DIGEST function, SECURITY-POLICY system handle

GENERATE-RANDOM-KEY function

Generates a pseudorandom (rather than a truly random) series of bytes to use as an encryption key, and returns the key as a RAW value.

Syntax

```
GENERATE-RANDOM-KEY
```

Notes

- You are responsible for generating, storing, and transporting this value.
- The size of the generated encryption key is determined by the cryptographic algorithm specified by the SYMMETRIC-ENCRYPTION-ALGORITHM attribute.
- The Alternate Random Number Generator (`-rand`) startup parameter setting has no effect on this function.

See also

GENERATE-PBE-KEY function, SECURITY-POLICY system handle

GENERATE-UUID function

Generates a universally unique identifier (UUID), as a 16-byte raw value.

Syntax

```
GENERATE-UUID
```

Example

The following code fragment illustrates how to use the GENERATE-UUID function:

```
DEFINE VARIABLE MyUUID as RAW.
DEFINE VARIABLE Base64UUID as CHARACTER.

MyUUID = GENERATE-UUID.
Base64UUID = BASE64-ENCODE(MyUUID).
```

You can use the GENERATE-UUID function with the BASE64-ENCODE function to generate a UUID and convert it to use in a Base64 character index. You can also remove the two trailing Base64 pad characters to reduce the size of the UUID. For example:

```
SUBSTRING(BASE64-ENCODE(GENERATE-UUID), 1, 22)
```

See also

BASE64-ENCODE function, GUID function

GET statement

Returns one record for a previously opened query.

Syntax

```
GET { FIRST | NEXT | PREV | LAST | CURRENT } query
    [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
    [ NO-WAIT ]
```

FIRST *query*

Finds the first record associated with the query. The query must have been previously opened in an OPEN QUERY statement. The order of the records is determined by the options specified in the Record phrase the OPEN QUERY statement.

NEXT *query*

Returns the first or next record associated with the query. The query must have been previously opened in an OPEN QUERY statement. The order of the records is determined by the options specified in the OPEN QUERY statement of the Record phrase.

PREV *query*

Returns the preceding or last record associated with the query. The query must have been previously opened in an OPEN QUERY statement. The order of the records is determined by the options specified in the OPEN QUERY statement of the Record phrase.

LAST *query*

Returns the last record associated with the query. The query must have been previously opened in an OPEN QUERY statement. The order of the records is determined by the options specified in the OPEN QUERY of the Record phrase.

CURRENT *query*

Refetches the current record or records associated with the query. The query must have been previously opened in an OPEN QUERY statement. If the query is a join, Progress returns the current record for all tables in the join.

SHARE-LOCK

Specifies that the record is share locked. Overrides the default locking of the OPEN QUERY statement. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. Overrides the default locking of the OPEN QUERY statement. This applies to all buffers in a join.

NO-LOCK

Specifies that no lock is applied to the record. Overrides the default locking of the OPEN QUERY statement. This applies to all buffers in a join.

NO-WAIT

Specifies that the GET statement returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the GET statement waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

Example

This procedure uses the GET statement to find customer orders:

r-getord.p

```
DEFINE QUERY cust-order FOR customer, order.

OPEN QUERY cust-order FOR EACH customer,
  EACH order OF customer.

GET FIRST cust-order.

DO WHILE AVAILABLE(customer):

  DISPLAY customer.cust-num customer.name
    WITH FRAME cust-info.
  DISPLAY order WITH FRAME order-info SIDE-LABELS.
  PAUSE.

  GET NEXT cust-order.
END.
```

In the example, the GET FIRST statement fetches the first customer record and the first order record for that customer. The GET NEXT statement fetches the next order record for the customer. If no more order records are found for the current customer, then the GET NEXT statement fetches the next customer and the first order record for that customer. If a customer has no orders, the GET statement skips that customer.

Notes

- The query must be opened with the OPEN QUERY statement before any records are fetched.
- A query that references more than one buffer defines a join. Each GET statement returns one set of records.
- If you execute a GET NEXT statement after the last record of the query has been fetched or you execute a GET PREV statement after the first record of the query has been fetched, the ERROR condition it **not** raised. However, you **can** use the AVAILABLE function to test whether a record was returned for the query fetch.
- If the query is positioned before the first record, GET NEXT acts the same as a GET FIRST; similarly, if the query is positioned beyond the last record, GET PREV acts the same as GET LAST.
- The GET LAST statement can be slow unless Progress has performed a presort or already returned the last record that satisfies the query, or you specify USE-INDEX for the query (or the query happens to only use one index). Also, GET LAST might be slow if the query involves an outer join.
- If you do not specify a lock type, Progress uses the lock type specified in the OPEN QUERY statement. If no lock type is specified in either the GET or OPEN QUERY statement, then the default Progress locking rules apply.
- If a GET CURRENT statement fails because of a lock conflict, Progress rereads the record with a NO-LOCK status.
- When a GET statement executes, any FIND triggers defined for the tables are executed.
- FIND triggers do not execute for a GET CURRENT statement.
- To upgrade the lock on only one table in a join, use the FIND CURRENT statement.

See also

AVAILABLE function, CLOSE QUERY statement, CURRENT-CHANGED function, CURRENT-RESULT-ROW function, DEFINE QUERY statement, FIND statement, FOR statement, LOCKED function, NUM-RESULTS function, OPEN QUERY statement, REPOSITION statement

GET-BITS function

Interprets one or more consecutive bits in an integer as a Progress INTEGER value and returns that value.

Syntax

```
GET-BITS( source , position , numbits )
```

source

A Progress integer variable.

position

A variable or expression that returns an integer. This parameter designates the position of the lowest-order bit of the bits that are to be interpreted as an integer. Bits are numbered from 1 through the length of an integer; with 1 being the low-order bit. If *position* is greater than the length of an INTEGER, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a runtime error.

numbits

The number of bits to examine when generating the return value. If *position* plus *numbits* is greater than the length of an integer plus 1, Progress generates a runtime error.

See also

PUT-BITS statement

GET-BYTE function

Returns the unsigned 1 byte value at the specified memory location as an INTEGER.

Syntax

```
GET-BYTE ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-BYTE returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a run-time error.

Examples

In this example, the RAW function goes to the customer field in the non-OpenEdge database. The GET-BYTE function accesses the first byte and stores the integer value of that byte in the variable i. The procedure then tests the value, if the integer value is 83 (the character code value for S), Progress displays the name.

r-rawget.p

```
/*You must connect to a non-PROGRESS database to run this procedure*/

DEFINE VARIABLE i AS INTEGER.

FOR EACH customer:
  i = GET-BYTE(RAW(name),1).
  IF i = 83
  THEN DISPLAY NAME.
END.
```

The next procedure sets up a MEMPTR region with a character string and uses the GET-BYTE function to display the character code value of each character in the string.

r-mptget.p

```
DEFINE VARIABLE mptr AS MEMPTR.
DEFINE VARIABLE cnt  AS INTEGER.

SET-SIZE(mptr) = LENGTH("DANIEL") + 1.
PUT-STRING(mptr, 1) = "DANIEL".

REPEAT cnt = 1 TO LENGTH("DANIEL"):
  DISPLAY GET-BYTE(mptr, cnt).
END.
```

Notes

- For more information on using the MEMPTR data type, see *OpenEdge Development: Programming Interfaces*.
- For more information on using the RAW data type, see *OpenEdge Development: Programming Interfaces*.
- You can use the alternative keyword GETBYTE instead of GET-BYTE.

See also

LENGTH function, PUT-BYTE statement, RAW function, RAW statement, SET-SIZE statement

GET-BYTE-ORDER function

Returns an integer indicating the byte order setting of a MEMPTR variable. This will be either the value provided by the last execution of SET-BYTE-ORDER with this MEMPTR variable, or HOST-BYTE-ORDER if SET-BYTE-ORDER has not been executed.

Syntax

```
GET-BYTE-ORDER( memptr )
```

memptr

An expression that returns a MEMPTR.

Note

GET-BYTE-ORDER never affects data currently in the MEMPTR. That is, it does not actually re-order the data.

See also

SET-BYTE-ORDER statement

GET-BYTES function

Returns the specified number of bytes, from the specified location, into a RAW or MEMPTR variable.

Syntax

```
GET-BYTES( source , position , numbytes )
```

source

An expression that evaluates to a RAW or MEMPTR value that indicates the source location. If *source* is the Unknown value (?), GET-BYTES returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position of the first byte to get. If *position* is greater than the length of *source*, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a runtime error.

numbytes

An integer value greater than 0 that indicates how many bytes to return as a RAW value. If *position* plus *numbytes* is greater than the size of *source*, Progress returns the Unknown value (?).

If the variable that accepts the returned data is a RAW variable and *numbytes* is greater than its length but less than or equal to 32K, Progress increases the size of the variable to *numbytes*.

If either the source location, *source*, or the variable that accepts the returned data is a RAW value, and *numbytes* is greater than 32K, Progress generates a runtime error.

If the variable that accepts the returned data is a MEMPTR variable and *numbytes* is greater than its length, Progress generates a runtime error.

See also

PUT-BYTES statement

GET-CODEPAGE function

The GET-CODEPAGE function returns the code page of a LONGCHAR variable or CLOB field.

Syntax

```
GET-CODEPAGE ( large-char-object )
```

large-char-object

The name of a LONGCHAR variable or CLOB field. If the specified LONGCHAR is empty and the code page was not fixed using the FIX-CODEPAGE function, Progress returns the Unknown value (?).

See also FIX-CODEPAGE function, IS-CODEPAGE-FIXED( ) function

GET-CODEPAGES function

The GET-CODEPAGES function returns a comma-delimited list of the code pages listed in `convmap.cp` or specified by the Conversion Map (-convmap) startup parameter for the current OpenEdge session.

Syntax

```
GET-CODEPAGES
```

Example

This procedure displays a list of the code pages available in memory for the current OpenEdge session and the collations available for each code page:

r-get.p

```
DEF VARIABLE code-page-list AS CHARACTER.
DEF VARIABLE collation-list AS CHARACTER.
DEF VARIABLE i AS INTEGER.
DEF VARIABLE j AS INTEGER.

code-page-list = GET-CODEPAGES.

REPEAT i = 1 TO NUM-ENTRIES(code-page-list):
  DISPLAY ENTRY(i,code-page-list) FORMAT "x(19)" COLUMN-LABEL "Code Page"
    WITH DOWN FRAME a.
  collation-list = GET-COLLATIONS(ENTRY(i,code-page-list)).
  REPEAT j = 1 TO NUM-ENTRIES(collation-list):
    DISPLAY ENTRY(j,collation-list) FORMAT "x(19)" COLUMN-LABEL "Collation"
      WITH DOWN FRAME a.
    DOWN WITH FRAME a.
  END.
END.
```

See also

GET-COLLATIONS function

GET-COLLATION function

The GET-COLLATION function returns the collation name for a CLOB field.

Syntax

```
GET-COLLATION ( clob-field )
```

clob-field

A CLOB field name.

GET-COLLATIONS function

The GET-COLLATIONS function returns a comma-delimited list of the collations either listed in `convmap.cp` or specified by the Conversion Map (`-convmap`) startup parameter for the specified code page.

Syntax

```
GET-COLLATIONS ( codepage )
```

codepage

A code page name. If there are no collations for the specified code page, Progress returns the Unknown value (?).

Example

This procedure displays a list of the code pages available in memory for the current OpenEdge session and the collations available for each code page:

r-get.p

```
DEF VARIABLE code-page-list AS CHARACTER.
DEF VARIABLE collation-list AS CHARACTER.
DEF VARIABLE i AS INTEGER.
DEF VARIABLE j AS INTEGER.

code-page-list = GET-CODEPAGES.

REPEAT i = 1 TO NUM-ENTRIES(code-page-list):
  DISPLAY ENTRY(i,code-page-list) FORMAT "x(19)" COLUMN-LABEL "Code Page"
    WITH DOWN FRAME a.
  collation-list = GET-COLLATIONS(ENTRY(i,code-page-list)).
  REPEAT j = 1 TO NUM-ENTRIES(collation-list):
    DISPLAY ENTRY(j,collation-list) FORMAT "x(19)" COLUMN-LABEL "Collation"
      WITH DOWN FRAME a.
    DOWN WITH FRAME a.
  END.
END.
```

See also

GET-CODEPAGES function

GET-DOUBLE function

Returns the 8-byte floating-point value at the specified memory location as a DECIMAL.

Syntax

```
GET-DOUBLE ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-DOUBLE returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a run-time error.

Examples

For examples of how to use the GET-DOUBLE function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. Progress does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR data type, see *OpenEdge Development: Programming Interfaces*.
- For more information on using the RAW data type, see *OpenEdge Development: Programming Interfaces*.

See also

LENGTH function, PUT-DOUBLE statement, RAW function, RAW statement, SET-SIZE statement

GET-FLOAT function

Returns the 4-byte floating-point value at the specified memory location as a DECIMAL.

Syntax

```
GET-FLOAT ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-FLOAT returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a run-time error.

Examples

For examples of how to use the GET-FLOAT function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. Progress does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR data type, see *OpenEdge Development: Programming Interfaces*.
- For more information on using the RAW data type, see *OpenEdge Development: Programming Interfaces*.

See also

LENGTH function, PUT-FLOAT statement, RAW function, RAW statement, SET-SIZE statement

GET-KEY-VALUE statement (Windows only)

Searches the current environment for a particular key and places its value into a particular data item.

Note: Does not apply to SpeedScript programming.

Syntax

```
GET-KEY-VALUE SECTION section-name
  KEY { key-name | DEFAULT }
  VALUE key-value
```

SECTION *section-name*

A CHARACTER expression that specifies the name of the section that contains the key of interest.

In initialization files, section names appear in square brackets([]). When you specify a section name in the GET-KEY-VALUE statement, omit the square brackets.

KEY *key-name*

A CHARACTER expression that specifies the name of the key of interest.

If you specify the Unknown value (?) or the empty string (""), GET-KEY-VALUE returns a comma-separated list of all keys in the section you specified.

DEFAULT

Tells GET-KEY-VALUE to use the default key of section *section-name.*

Some applications store data in the registry under the default key of a section. This option lets you retrieve this data. For an example, see the EXAMPLES section of this entry.

This option applies only to the registry and not to initialization files.

VALUE *key-value*

The name of a CHARACTER variable to hold the value of the key of interest.

Examples If the current environment resides in the registry, the following example:

1. Searches the current environment for the subkey MYSECTION.
2. Searches MYSECTION for the value name MYKEY.
3. Assigns the value of MYKEY to the variable MYVARIABLE.

If the current environment resides in an initialization file, the following example:

1. Searches the section MYSECTION for the key MYKEY.
2. Assigns the value of MYKEY to the variable MYVARIABLE as shown in the following example:

```
GET-KEY-VALUE SECTION "MYSECTION" KEY "MYKEY" VALUE MYVARIABLE
```

If the current environment is the registry, the following example:

1. Searches the current environment for the key MYKEY.
2. Assigns the value of MYKEY to the variable MYVARIABLE.

If the current environment resides in an initialization file, the following example returns a comma-separated list of all section names in the initialization file:

```
GET-KEY-VALUE SECTION "" KEY "MYKEY" VALUE MYVARIABLE
```

If the current environment resides in the registry, the following examples:

1. Search the current environment for the subkey MYSECTION.
2. Return a comma-separated list of all value names in MYSECTION.

If the current environment resides in an initialization file, the following examples:

1. Search the current environment for the section MYSECTION.
2. Return a comma-separated list of all key names in MYSECTION.

```
GET-KEY-VALUE SECTION "MYSECTION" KEY "" VALUE MYVARIABLE
```

```
GET-KEY-VALUE SECTION "MYSECTION" KEY "?" VALUE MYVARIABLE
```

If the current environment resides in the registry, the following examples return a comma-separated list of subkeys under the current environment location and all value names directly under the current environment location. The delimiter @value@ separates the subkey names from the value names.

If the current environment resides in an initialization file, the following examples return a comma-separated list of all section names in the initialization file:

```
GET-KEY-VALUE SECTION "" KEY "" VALUE MYVARIABLE
```

```
GET-KEY-VALUE SECTION "" KEY "?" VALUE MYVARIABLE
```

```
GET-KEY-VALUE SECTION "?" KEY "" VALUE MYVARIABLE
```

```
GET-KEY-VALUE SECTION "?" KEY "?" VALUE MYVARIABLE
```

If the current environment resides in the registry, the following example:

1. Searches the current environment for the subkey MYAPP.
2. Assigns the value of the default key under MYAPP to the variable MYVARIABLE.

If the current environment resides in an initialization file, the following example returns an error.

```
GET-KEY-VALUE SECTION "MYAPP" KEY DEFAULT VALUE MYVARIABLE
```

Notes

- Environments typically consist of sections, each of which can contain keys, each of which consists of a name and a value. A typical section name is COLORS. A typical key within this section consists of the name “COLOR16" and the value 255,255,0. This key attaches this particular name to this particular color. (The value represents a color specification using the red-green-blue color-naming scheme.)

  The current environment might be the registry or an initialization file. The registry consists of sections called keys and subkeys arranged in a hierarchy. Keys and subkeys contain value entries, each of which consists of a value name and value data. Initialization files, by contrast, consist of a single level of sections. Sections contain entries, each of which consists of a name, an equals sign (=), and a value.

  For more information on environments, see the chapter on colors and fonts in *OpenEdge Development: Programming Interfaces*.

- The current environment is either the default environment, the **startup environment** (an environment that a startup parameter specified), or an application environment that the LOAD statement loaded and that the USE statement made current.
- If you unload the current environment (using the UNLOAD statement) and then use the GET-KEY-VALUE statement, you access the startup environment.

See also LOAD statement, PUT-KEY-VALUE statement, UNLOAD statement, USE statement

GET-LONG function

Returns the signed 32-bit value at the specified memory location as an INTEGER.

Syntax

```
GET-LONG ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-LONG returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a runtime error.

Examples

For examples of how to use the GET-LONG function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. Progress does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR data type, see *OpenEdge Development: Programming Interfaces*.
- For more information on using the RAW data type, see *OpenEdge Development: Programming Interfaces*.

See also

LENGTH function, PUT-LONG statement, RAW function, RAW statement, SET-SIZE statement

GET-POINTER-VALUE function

Returns (as an INTEGER value) the address of (or pointer to) the memory region associated with the specified MEMPTR variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
GET-POINTER-VALUE ( memptr-var )
```

memptr-var

A reference to a variable defined as MEMPTR. If the variable is uninitialized (has no associated memory region), the function returns 0.

Example

This function is particularly useful when building a structure in an MEMPTR region that references other MEMPTR regions. It allows you to obtain the pointer to one MEMPTR region and store it in the structure you create in another MEMPTR region. The following example allocates three memory regions-for a BITMAPINFO structure, a BITMAPINFOHEADER structure, and an RGB color array. It then uses the GET-POINTER-VALUE function together with the PUT-LONG statement to store pointers to the BITMAPINFOHEADER structure and an RGB color array in the BITMAPINFO structure. These structures describe a device-independent bitmap for Windows dynamic link library (DLL) routines. For more information on these bitmap structures, see your Windows Software Development Kit documentation.

r-ptrval.p

```
DEFINE VARIABLE bitmapinfo       AS MEMPTR.
DEFINE VARIABLE bitmapinfoheader AS MEMPTR.
DEFINE VARIABLE RGBcolors        AS MEMPTR.

SET-SIZE(bitmapinfo) = 4        /* Pointer to bitmapinfoheader */
                     + 4        /* Pointer to RGBcolors        */
                       .
SET-SIZE(bitmapinfoheader) = 4  /* biSize          */
                           + 4  /* biWidth         */
                           + 4  /* biHeight        */
                           + 2  /* biPlanes        */
                           + 2  /* biBitCount      */
                           + 4  /* biCompression   */
                           + 4  /* biSizeImage     */
                           + 4  /* biXpelsPerMeter */
                           + 4  /* biYPelsPerMeter */
                           + 4  /* biClrUsed       */
                           + 4  /* biClrImportant  */
                             .
SET-SIZE(RGBcolors) = 16 * 4.   /* Array for 16 RGB color values */

/* Initialize pointers to bit map info header and RGB color array */

PUT-LONG(bitmapinfo,1) = GET-POINTER-VALUE(bitmapinfoheader).
PUT-LONG(bitmapinfo,5) = GET-POINTER-VALUE(RGBcolors).
```

Note: Before using structures such as these, you must initialize them according to your DLL requirements. For example, the biBitCount segment of the bitmapinfoheader must be set to 4 to specify the number of possible colors available in the RGB color array (16).

Notes

- MEMPTR structures are initialized using the SET-SIZE statement.
- For more information on using the MEMPTR data type, see *OpenEdge Development: Programming Interfaces*.

See also

PUT-LONG statement, SET-SIZE statement

GET-SHORT function

Returns the signed 16-bit value at the specified memory location as an INTEGER.

Syntax

```
GET-SHORT ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-SHORT returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a run-time error.

Examples

For examples of how to use the GET-SHORT function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. Progress does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR data type, see the chapter on Windows dynamic link libraries (DLLs) in *OpenEdge Development: Programming Interfaces*.
- For more information on using the RAW data type, see *OpenEdge Development: Programming Interfaces*.

See also

LENGTH function, PUT-SHORT statement, RAW function, RAW statement, SET-SIZE statement

GET-SIZE function

Returns (as an INTEGER value) the allocated byte size of the memory region associated with the specified MEMPTR variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
GET-SIZE ( memptr-var )
```

memptr-var

A MEMPTR variable. If the variable is uninitialized (has no associated memory region), the function returns 0.

Example

The following example allocates three memory regions-for a BITMAPINFO structure, a BITMAPINFOHEADER structure, and an RGB color array. It then displays the allocation size for each region. These structures describe a device-independent bitmap for Windows dynamic link library (DLL) routines. For more information on these bitmap structures, see your Windows Software Development Kit documentation.

r-getsiz.p

```
DEFINE VARIABLE bitmapinfo       AS MEMPTR.
DEFINE VARIABLE bitmapinfoheader AS MEMPTR.
DEFINE VARIABLE RGBcolors        AS MEMPTR.

SET-SIZE(bitmapinfo) = 4        /* Pointer to bitmapinfoheader */
                     + 4        /* Pointer to RGBcolors        */
                       .
SET-SIZE(bitmapinfoheader) = 4  /* biSize          */
                           + 4  /* biWidth         */
                           + 4  /* biHeight        */
                           + 2  /* biPlanes        */
                           + 2  /* biBitCount      */
                           + 4  /* biCompression   */
                           + 4  /* biSizeImage     */
                           + 4  /* biXpelsPerMeter */
                           + 4  /* biYPelsPerMeter */
                           + 4  /* biClrUsed       */
                           + 4  /* biClrImportant  */
                            .
SET-SIZE(RGBcolors) = 16 * 4.   /* Array for 16 RGB color values */

DISPLAY
   GET-SIZE(bitmapinfo)
      LABEL "Bitmap info structure size" COLON 30 SKIP
   GET-SIZE(bitmapinfoheader)
      LABEL "Bitmap header structure size" COLON 30 SKIP
   GET-SIZE(RGBcolors)
      LABEL "Bitmap colors array size" COLON 30
WITH SIDE-LABELS.
```

Notes

- To return a memory size greater than 0, the MEMPTR variable must be fully initialized, not just pre-initialized by a DLL or UNIX shared library routine.
- MEMPTR structures are initialized using the SET-SIZE statement.
- For more information on using the MEMPTR data type, see *OpenEdge Development: Programming Interfaces*.

See also SET-SIZE statement

GET-STRING function

Returns the null-terminated character string at the specified memory location as a CHARACTER value (not including the null terminator) or the number of bytes specified starting from the specified memory location as a CHARACTER value.

Syntax

```
GET-STRING ( source , position [ , numbytes ] )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-STRING returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a run-time error.

numbytes

An integer value greater than 0 that indicates how many bytes to convert into the CHARACTER value that is returned. If *position* plus *numbytes* is greater than the length of *source*, Progress returns the Unknown value (?). If *numbytes* is not specified, or is -1, GET-STRING( ) returns all bytes until it encounters a NULL value.

Examples

For examples of how to use the GET-STRING function, see the GET-BYTE function reference entry.

Notes

- For more information on using the MEMPTR data type, see *OpenEdge Development: Programming Interfaces*.
- For more information on using the RAW data type, see *OpenEdge Development: Programming Interfaces*.

See also

LENGTH function, PUT-STRING statement, RAW function, RAW statement, SET-SIZE statement

GET-UNSIGNED-SHORT function

Returns the unsigned 16-bit value at the specified memory location as an INTEGER.

Syntax

```
GET-UNSIGNED-SHORT ( source , position )
```

source

A function or variable that returns a RAW or MEMPTR value. If *source* is the Unknown value (?), GET-UNSIGNED-SHORT returns the Unknown value (?).

position

An integer value greater than 0 that indicates the byte position where you want to find the information. If *position* is greater than the length of *source*, Progress returns the Unknown value (?). If *position* is less than 1, Progress generates a run-time error.

Examples

For examples of how to use the GET-UNSIGNED-SHORT function, see the GET-BYTE function reference entry.

Notes

- This function supports byte-swapping only if *source* is a MEMPTR data type. The function will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately before interpreting them. Progress does not swap the bytes in the MEMPTR's memory, but does the byte-swap as it creates the return value.
- For more information on using the MEMPTR data type, see the chapter on Dynamic Link Libraries (DLLs) in *OpenEdge Development: Programming Interfaces*.
- For more information on using the RAW data type, see *OpenEdge Development: Programming Interfaces*.

See also

LENGTH function, PUT-SHORT statement, RAW function, RAW statement, SET-SIZE statement

GO-PENDING function

Returns a TRUE value if, within an EDITING phrase, an APPLY statement results in a GO action. The GO action is deferred until the end of the EDITING phrase.

This function is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
GO-PENDING
```

Example

The `r-gopend.p` procedure lets you update some of the fields in each customer record. If you press **GO** when the value in the current balance field is greater than the balance in the credit-limit field, the UPDATE statement does not end. Instead, it continues prompting you for input until you correct the problem and then press **GO**.

r-gopend.p

```
REPEAT:
   PROMPT-FOR customer.cust-num.
   FIND customer USING cust-num.
   UPDATE
      name address city st SKIP
      credit-limit balance WITH 1 COLUMN
      EDITING:
        READKEY.
        APPLY LASTKEY.
        IF GO-PENDING AND INPUT balance > INPUT credit-limit THEN DO:
           MESSAGE "The current unpaid balance exceeds the credit limit.".
        NEXT.
      END.
   END.
END.
```

See also

APPLY statement, EDITING phrase

GT or > operator

Returns a TRUE value if the first of two expressions is greater than the second expression.

Syntax

```
expression { GT | > } expression
```

expression

A constant, field name, variable name, or any combination of these. The expressions on either side of the GT or > must be of the same data type, although one might be integer and the other decimal.

Example

This procedure lists all items that have a negative on-hand quantity or more than 90% of the on-hand inventory currently allocated:

r-gt.p

```
FOR EACH item:
  IF allocated > 0
  THEN IF (on-hand <= 0) OR
     (allocated / on-hand > .9)
  THEN DISPLAY item-num item-name on-hand allocated.
END.
```

Notes

- By default, Progress uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (-cpcoll) startup parameter take precedence over a collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. If you do not specify a collation with the -cpcoll startup parameter, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the default name "basic" (which might or might not exist in the convmap.cp file).
- If either of the expressions is the Unknown value (?), then the result is the Unknown value (?); if both of the expressions are the Unknown value (?), then the result is FALSE.

- You can compare character strings with GT. Most character comparisons are case insensitive in Progress. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *expression* is a field or variable defined as case sensitive, the comparison is case sensitive and “Smith” does not equal “smith”.

- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b). Note also that in character code uppercase A is less than [, \ , ^ , _, and ’ , but lowercase a is greater than these.

- You can use GT to compare DATE, DATETIME, and DATETIME-TZ data. The data type that contains less information (that is, a DATE value contains less information than a DATETIME value, and a DATETIME value contains less information than a DATETIME-TZ value) is converted to the data type with more information by setting the time value to midnight, and the time zone value to the session's time zone (when the data type does not contain the time or time zone). Comparisons with DATETIME-TZ data are based on Coordinated Universal Time (UTC) date and time.

- You can use GT to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or Progress raises a run-time error.

- You cannot use GT to compare one CLOB field to another.

GUID function

Converts a universally unique identifier (UUID) value into a globally unique identifier (GUID) value. This function returns a GUID as a 36-character string value consisting of 32 hexadecimal digits (0 through 9 and A through F) and 4 hyphens formatted as follows (where X is a hexadecimal digit): XXXXXXXX–XXXX–XXXX–XXXX–XXXXXXXXXXXX

Syntax

```
GUID( [ UUID ] )
```

UUID

An optional 16-byte raw UUID value to be converted. If the specified UUID is not exactly 16 bytes in length, Progress returns the Unknown value (?). If not specified, Progress generates a UUID and then converts it into a GUID.

Example

The following code fragment illustrates how to use the GUID function:

```
DEFINE VARIABLE MyUUID as RAW.
DEFINE VARIABLE vGUID as CHARACTER.

MyUUID = GENERATE-UUID.
vGUID = GUID(MyUUID).
```

See also

GENERATE-UUID function

HEX-DECODE function

Converts a character string consisting of an even number of hexadecimal digits (0 through 9 and A through F) into a RAW value.

Syntax

```
HEX-DECODE( expression )
```

expression

A character expression containing the value you want to convert. If the expression does not contain an even number of hexadecimal digits, or it is the Unknown value (?), the result is the Unknown value (?). If the expression is a zero-length value, the result is a zero-length value.

Example

The following code fragment illustrates how to use the HEX-DECODE function:

```
DEFINE VARIABLE vRaw as RAW.

vRaw = HEX-DECODE(HEX-ENCODE(GENERATE-UUID)).
```

See also

HEX-ENCODE function

HEX-ENCODE function

Converts a RAW value into a character string consisting of an even number of hexadecimal digits (0 through 9 and A through F).

Syntax

```
HEX-ENCODE( expression )
```

expression

A RAW expression containing the value you want to convert. If the expression is the Unknown value (?), the result is the Unknown value (?). If the expression is a zero-length value, the result is a zero-length value.

Example

The following code fragment illustrates how to use the HEX-ENCODE function:

```
DEFINE VARIABLE MyUUID as RAW.
DEFINE VARIABLE vChar as CHARACTER.

MyUUID = GENERATE-UUID.
vChar = HEX-ENCODE(MyUUID).
```

See also

HEX-DECODE function

HIDE statement

Makes a widget invisible (sets its VISIBLE attribute to FALSE), or clears the message area for a window, or hides all widgets and clears messages in a window.

Note: Does not apply to SpeedScript programming.

Syntax

```
HIDE [ STREAM stream ]
  [ widget-phrase | MESSAGE | ALL ]
  [ NO-PAUSE ]
  [ IN WINDOW window ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream.

widget-phrase

The widget you want to hide. You can hide windows, frames, and field-level widgets. You cannot hide menus. If you do not use this option or the MESSAGE or ALL options, HIDE hides the default frame for the block that contains the HIDE statement.

MESSAGE

Hides all messages displayed in the message area for the specified window. If you use the PUT SCREEN statement to display data in the message area, the HIDE MESSAGE statement does not necessarily hide that data.

ALL

Hides all widgets in the window and clears the message area for the window.

NO-PAUSE

Does not pause before hiding. Ordinarily, if data has been displayed, but there have been no data entry operations or pauses, Progress prompts you to press **SPACEBAR** to continue before hiding the widget.

IN WINDOW *window*

Specifies which window the HIDE statement acts on. The value *window* must evaluate to the handle of a window. If you do not use the IN WINDOW option, the current window is assumed.

Example

The following example uses the HIDE statement to hide selected frames. The DISPLAY statements redisplays the frames when the loop iterates.

r-hide.p

```
DEFINE VARIABLE selection AS INTEGER FORM "9".

FORM   "Please Make A Selection:" SKIP(2)
       "    1. Hide Frame A.    " SKIP
       "    2. Hide Frame B.    " SKIP
       "    3. Hide All.        " SKIP
       "    4. Hide This Frame  " SKIP
       "    5. Exit             " SKIP(2)
       WITH FRAME X NO-LABELS.

REPEAT:
    VIEW FRAME x.
    DISPLAY "This is frame A."
      WITH FRAME a ROW 1 COLUMN 60.
    DISPLAY "This is frame B."
      WITH FRAME b ROW 16 COLUMN 10 4 DOWN.
    MESSAGE "Make your selection!".
    UPDATE "Selection: " selection
           VALIDATE(0 < selection AND selection < 7,
                    "Invalid selection")  AUTO-RETURN
           WITH FRAME x.

    IF selection = 1 THEN HIDE FRAME a.
    ELSE IF selection = 2 THEN HIDE FRAME b.
    ELSE IF selection = 3 THEN HIDE ALL.
    ELSE IF selection = 4 THEN HIDE FRAME x.
    ELSE IF selection = 5 THEN LEAVE.
    PAUSE.
END.
```

Notes

- When a block iterates, any display frame that is scoped to the block or to a nested block is tagged for hiding (unless you have used the NO-HIDE option in the Frame phrase), but is not hidden. Then, the first frame activity of the next iteration (a DISPLAY, INSERT, PROMPT-FOR, SET, VIEW, or UPDATE statement) for a frame scoped to the block or to a nested block causes all tagged frames to be hidden.

  The frame associated with that first frame activity is not hidden because it would be redisplayed immediately. This improves display time. When a block ends, Progress removes the hide tags from all the frames scoped to that block or to nested blocks.

- Frames displayed by procedures within a block or within a nested block are treated the same as other frames in a nested block.

- When Progress displays a frame and there is not enough room in the window, it automatically hides one or more frames. Frames are hidden one at a time, starting with the lowest frame in the window, until there is room to fit the new frame.

- It is more efficient to HIDE ALL than to HIDE each frame individually.

- If you hide a PAGE-TOP or PAGE-BOTTOM frame, it is removed from the list of active frames for printing at the top or bottom of each page.

- If you are working in a PAGE-TOP or PAGE-BOTTOM frame, use the VIEW or DISPLAY statement to activate that frame. The VIEW statement does not display a PAGE-TOP or PAGE-BOTTOM frame. It activates the frame so that when a new page begins or ends, Progress displays the frame. If you use the HIDE statement to hide a PAGE-TOP or PAGE-BOTTOM frame, Progress deactivates that frame so that it can no longer be displayed unless it is reactivated with a VIEW or DISPLAY statement.

- If output is not directed to the terminal, HIDE has no effect on the terminal display.

- In batch mode, the HIDE statement produces an error. If you want to remove the contents of a frame, use the CLEAR statement instead.

- You can use HIDE MESSAGE to hide a message.

- If you invoke the HIDE statement for a field-level widget or child frame, the HIDDEN attribute of the specified field-level widget or child frame is also set to TRUE. However, if you invoke the HIDE statement for a child window, the HIDDEN attribute of the child window is unaffected.
- When you HIDE a visible window, any visible descendant windows are hidden also (including iconified descendants), but any visible ancestor windows remain unaffected. However, if you HIDE a window whose HIDDEN attribute is currently set to TRUE, its descendant windows remain unaffected.

See also CLEAR statement, VIEW statement, Widget phrase

IF...THEN...ELSE function

Evaluates one of two expressions, depending on the value of a specified condition.

Syntax

```
IF condition THEN expression1 ELSE expression2
```

condition

An expression whose value is logical (TRUE or FALSE).

expression1

A constant, field name, variable name, or expression. If the *condition* is TRUE, then the function returns this value.

expression2

A constant, field name, variable name, or expression whose value is of a data type that is compatible with the data type of *expression1*. If the *condition* is FALSE or the Unknown value (?), then the function returns this value.

Example

You can use the IF . . . THEN . . . ELSE function when you want to sort records in an unusual order. In this example, the customers are sorted so that those with a balance greater than $10,000 appear first, then those with balances between $1,000 and $10,000, and finally those with balance of $1,000 or less.

r-ifelsf.p

```
FOR EACH customer
  BY IF balance > 10000 THEN 1
  ELSE (IF balance > 1000 THEN 2 ELSE 3) BY sales-rep:
  DISPLAY sales-rep balance name.
END.
```

IF...THEN...ELSE statement

Makes the execution of a statement or block of statements conditional. If the value of the expression following the IF statement is TRUE, Progress processes the statements following the THEN statement. Otherwise, Progress processes the statements following the ELSE statement.

Syntax

```
IF expression THEN { block | statement }
  [ ELSE { block | statement } ]
```

expression

A constant, field name, variable name, or expression whose value is logical (TRUE or FALSE). The expression can include comparisons, logical operators, and parentheses.

THEN

Describes the block statement to process if the *expression* is TRUE.

block

The block statement that contains the code you want to process if *expression* is TRUE. See the DO statement, FOR statement, and REPEAT statement reference entries for more information. If you do not start a block, you can process just one statement after the IF keyword or the ELSE keyword.

Any block or blocks you use in an IF . . . THEN . . . ELSE statement can contain other blocks or other IF . . . THEN . . . ELSE statements.

statement

A single Progress statement. The *statement* can be another IF . . . THEN . . . ELSE statement. If you want to use more than one statement, enclose those statements in a DO, FOR EACH, or REPEAT block.

ELSE

Describes the block statement to process if the *expression* is FALSE or the Unknown value (?). The ELSE option is not required.

Example

The r-ifelss.p procedure creates a report in a file that lists customers whose orders have been shipped but who have not paid for those orders:

r-ifelss.p

```
DEFINE VARIABLE ans AS LOGICAL.
DEFINE STREAM due.

OUTPUT STREAM due TO ovrdue.lst.
   DISPLAY STREAM due
           "Orders shipped but still unpaid as of" TODAY SKIP(2)
           WITH NO-BOX NO-LABELS CENTERED FRAME hdr PAGE-TOP.

FOR EACH order WITH FRAME oinfo:
    FIND customer OF order NO-LOCK.
    DISPLAY order-num name order-date promise-date ship-date.
    IF ship-date = ? THEN DO:
        IF promise-date = ? THEN DO:
           MESSAGE "Please update the promise date.".
           REPEAT WHILE promise-date = ?:
              UPDATE promise-date WITH FRAME oinfo.
           END.
        END.
        ans = FALSE.
        MESSAGE "Has this order been shipped?" UPDATE ans.
        IF ans
        THEN REPEAT WHILE ship-date = ?:
           UPDATE ship-date WITH FRAME oinfo.
        END.
    END.
    ELSE DO:
        ans = TRUE.
        MESSAGE "Has this order been paid?" UPDATE ans.
        IF NOT ans THEN DO:
            DISPLAY STREAM due order-num TO 14 name AT 18
                    order-date AT 42 ship-date AT 54
                    WITH NO-BOX DOWN FRAME unpaid.
        END.
    END.
END.
OUTPUT STREAM due CLOSE.
```

First, the procedure writes report headers to the ovrdue.lst file. Next, the outer FOR EACH block reads each of the orders using a DISPLAY statement to display information on each order. If there are no values in the ship-date and promise-date fields, the procedure prompts you to enter a promise date. The procedure then prompts if the order has been shipped. If it has, supply a ship date.

If there is a ship date and a promise date for an order, the procedure prompts if the order has been paid for. If not, the procedure displays the order information to the file.

Image phrase

Specifies the file in which an image is stored and the dimensions of the image.

Note: Does not apply to SpeedScript programming.

Syntax

```
FILE name
  [ { IMAGE-SIZE | IMAGE-SIZE-CHARS | IMAGE-SIZE-PIXELS }
      width BY height
  ]
  [ FROM { X n Y n | ROW n COLUMN n } ]
```

```
{ IMAGE-SIZE | IMAGE-SIZE-CHARS | IMAGE-SIZE-PIXELS }
  width BY height
  [ FROM { X n Y n | ROW n COLUMN n } ]
```

FILE *name*

A character expression that specifies the name of an operating system file that contains an image. If you do not specify a full pathname, Progress searches your PROPATH for the file. If you do not supply a suffix, Progress searches for files with the extension .bmp, .ico, or .cur in Windows. The image contained within the file must be in a format that is appropriate for the target platform. The file is not read until the image is displayed.

[IMAGE-SIZE | IMAGE-SIZE-CHARS]

Specifies that the unit of measure when reading the image is characters.

IMAGE-SIZE-PIXELS

Specifies that the unit of measure when reading the image is pixels.

width

Specifies the width of the image. The value *width* must be an integer constant. If the image is larger than the size you specify, Progress crops the image to the specified size.

height

Specifies the height of the image. The value *height* must be an integer constant. If the image is larger than the size you specify, Progress crops the image to the specified size.

FROM { X *n* Y *n* | ROW *n* COL *n* }

Two integer constants (*n*) that specify the offset inside the image file where Progress starts reading the image. If you specify X and Y, the offset is measured in pixels; if you specify ROW and COL, the offset is measured in characters.

Example

See the DEFINE IMAGE statement reference entry for an example.

Notes

- Use one of the image size options in conjunction with the FILE option to make a compile-time association between the image file and the image widget; the image file does not have to exist at this point.
- Use one of the image size options without the FILE option to create an image widget that is not associated with an image file at compile time. You can then make the association at run time.
- Use the FILE option without one of the image size options if you do not know the size of the image and want Progress to determine the size at compile time. If you do this, Progress uses the entire image. Also note that the image file must exist or a compiler error will occur.
- In Windows, you can specify a URL pathname. If you do not specify a fully-qualified URL, Progress searches in the PROPATH for the file. Valid URL protocols include HTTP and HTTPS.

Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, Progress continues searching with the next PROPATH entry.

- The following image file formats are currently supported for use on button and image widgets:

File extension	Image file type
.bmp	Windows bitmap
.cal	Computer-aided Acquisition and Logistics Support
.clp	Microsoft Windows Clipboard
.cut	Halo CUT
.dcx	Intel FAX format
.dib	Windows device-independent bitmap
.eps	Encapsulated PostScript
.gif [1]	Graphics Interchange Format
.ica	IBM IOCA
.ico	Microsoft Icon File format
.iff	Amiga IFF
.img	GEM bitmap
.jbig	Joint Bi-level Image Experts Group
.jpg	JPEG
.lv	LaserView
.mac	Macintosh MacPaint
.msp	Microsoft Windows Paint
.pcd	Kodak Photo CD
.pct	Macintosh PICT
.pcx	PC Paintbrush

.png	GIF (Graphics Interchange Format) replacement
.psd	Adobe Photoshop
.ras	Sun Raster (1-, 8-, 24-, or 32-bit Standard, BGR, RGB, and byte encoded)
.tga	TARGA
.tif	Tag image file format
.wbmp	Windows bitmap for wireless devices
.wmf	Windows metafiles
.wpg	WordPerfect graphics
.xbm (also .bm)	X bitmap
.xpm	Pixmap
.xwd	UNIX X Window Dump File format

[1] Animation in .gif files is not supported.

See also DEFINE BUTTON statement, DEFINE IMAGE statement, FORM statement

IMPORT statement

Reads a line from an input file that might have been created by EXPORT.

Syntax

```
IMPORT [ STREAM stream ]
  {    [ DELIMITER character ] { field | ^ } ...
     | [ DELIMITER character ] record [ EXCEPT field  ... ]
     | UNFORMATTED field
  }
  [ NO-LOBS ]
  [ NO-ERROR ]
```

```
IMPORT [ STREAM stream ] { memptr | longchar }
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream.

DELIMITER *character*

The character used as a delimiter between field values in the file. The *character* parameter must be a quoted **single** character. The default is a space character.

If you specify more than one character as a delimiter, Progress uses the first character as the delimiter.

field

The name of a field or variable to which you are importing data. The field or variable must have either the CHARACTER or RAW data type. If the data type is RAW, the IMPORT statement reads enough characters to fill the current length of the variable. If not enough characters are available to fill the current length, the length is reset to the number of characters read.

^

Use a caret (^) to skip a data value in each input line when input is being read from a file.

record

The name of a record buffer. All of the fields in the record are processed exactly as if you had named each of them individually. The record you name must contain at least one field. To use IMPORT with a record in a table defined for multiple databases, qualify the record's table name with the database name. See the Record phrase reference entry for more information.

EXCEPT *field*

Tells Progress to import all the fields except those listed in the EXCEPT phrase.

UNFORMATTED *field*

Treats each line of the input file as a single string value. In this case, the *field* parameter must be a single CHARACTER or RAW field or variable. You can use this option to read text files one line at a time.

Use this option on a RAW variable to import binary data that was not exported to the file as RAW data.

NO-LOBS

Directs Progress to ignore large object data when importing records that contain BLOB or CLOB fields.

NO-ERROR

Suppresses any errors that occur in the attempt to import the text. After the IMPORT statement completes, you can check the ERROR-STATUS system handle for information on errors that have occurred.

memptr

A variable of data type MEMPTR that contains the imported text. The IMPORT statement my contain a MEMPTR in its field list as long as it is the only field in the list.

longchar

A variable of data type LONGCHAR that contains the imported text. The IMPORT statement may contain a LONGCHAR in its field list as long as the LONGCHAR is the only field in the list and is the result of an EXPORT statement.

Examples

This procedure takes the data in file `customer.d` and enters it into the OpenEdge database table customer. The procedure uses the DISABLE TRIGGERS statement to stop Progress from executing any triggers for the CREATE, WRITE, and ASSIGN events when loading the data.

Note: The imported files, `customer.d` and `custdump2`, in the next two examples are created by running the example programs under EXPORT.

r-imprt.p

```
INPUT FROM customer.d.

DISABLE TRIGGERS FOR LOAD OF customer.

REPEAT:
    CREATE customer.
    IMPORT customer.
END.

INPUT CLOSE.
```

If the file uses a delimiter other than a space to separate fields, use the DELIMITER option of the IMPORT statement.

r-cstin.p

```
DEFINE VARIABLE cnum     LIKE customer.cust-num.
DEFINE VARIABLE cname    LIKE customer.name.
DEFINE VARIABLE cmax     LIKE customer.credit-limit.

INPUT FROM custdump2.

FOR EACH customer:
   IMPORT DELIMITER ";" cnum cname cmax.
   DISPLAY cnum cname cmax.
END.

INPUT CLOSE.
```

You can use the UNFORMATTED option to read the contents of a standard text file. For example, the following procedure reads and displays the contents of the hello file:

r-hello.p

```
DEFINE VARIABLE text-string AS CHARACTER FORMAT "x(76)".

INPUT FROM VALUE(SEARCH("hello")).

DO WHILE TRUE:
   IMPORT UNFORMATTED text-string.
   DISPLAY text-string WITH DOWN FRAME x.
   DOWN WITH FRAME x.
END.

INPUT CLOSE.
```

In the MEMPTR version of the IMPORT statement, the MEMPTR must be pre-allocated to the size needed for reading. To get the length to read for an imported file, use the FILE_INFO system handle and the SET-SIZE statement as follows:

r-impmem.p

```
DEFINE VARIABLE bb AS MEMPTR.
FILE-INFO:FILE-NAME = "big.in".
SET-SIZE(bb) = FILE-INFO:FILE-SIZE.
INPUT FROM "big.in" BINARY NO-CONVERT.
IMPORT bb.
INPUT CLOSE.
```

Notes

- The IMPORT statement must follow a statement that redirects the input source (usually an INPUT FROM statement). You cannot use the IMPORT statement to read data from the screen.
- If you do not use the UNFORMATTED option, the data in the input stream must be in a **standard format** to be read back into Progress. You must enclose all character fields in quotes ("") if they contain any delimiter characters. If you want to import any quotes contained in the data, replace them with two quotes (" "). You must display the Unknown value (?) as an unquoted question mark.

- If an input data line contains an unquoted hyphen in place of a data value, then the corresponding field is skipped, as it is in UPDATE. If you specify a hyphen (-) as the delimiter character, all hyphens are treated as delimiters. If you use the UNFORMATTED option, the hyphen is treated the same as any other character.

- A period (.) on a line by itself is treated as an end-of-file indicator. The ENDKEY is applied, but the file or stream remains open for input.

- Data read in with IMPORT is not restricted by frame-related format statements, as is data read in by SET or UPDATE. Since IMPORT does not have to validate the input stream, it is faster than SET or UPDATE.

- When importing records that contain a BLOB or CLOB field, Progress uses the value stored in the BLOB or CLOB field of the exported record to determine whether or not the exported record has an associated object data file to import. If the BLOB or CLOB field in the exported record contains the Unknown value (?), Progress stores the Unknown value (?) in the BLOB or CLOB field of the new or updated record. If the BLOB or CLOB field in the exported record contains a filename, Progress imports the associated object data. If an updated record already has object data associated with it, Progress deletes that object data before importing the new object data.

  Progress raises the ERROR condition if an object data file cannot be found or read.

- Use the `NO-LOBS` option with the IMPORT statement to ignore large object data when importing records that contain BLOB or CLOB fields. More specifically:

  – When you import an exported record into a new record, and the BLOB or CLOB field of the exported record contains either the Unknown value (?) or a filename, Progress sets the value of the BLOB or CLOB field in the newly imported record to the Unknown value (?); Progress does not create any object data.

  – When you import an exported record as an update to an existing record, and the BLOB or CLOB field of the exported record contains either the Unknown value (?) or a filename, Progress does not change the value of the BLOB or CLOB field in the existing record and neither creates nor overwrites object data.

- The IMPORT statement reads large object data files from the directory specified as the input data source in the INPUT FROM statement, by default. You can use the `LOB-DIR` option on the INPUT FROM statement to specify the directory from which the IMPORT statement reads BLOB and CLOB data files.

- IMPORT is sensitive to the Date Format (-d), Century (-yy), and European Numeric Format (-E) parameters. When loading data with the IMPORT statement, use the same settings that you used with the EXPORT statement.
- When importing DATETIME and DATETIME-TZ data, the data format must be fixed and must conform to the ISO 8601 standard for date/time representations (`YYYY-MM-DDTHH:MM:SS.SSS+HH:MM`). For DATETIME, there is no time zone offset.
- Progress interprets the null character as a terminator.
- The UNFORMATTED option forces IMPORT to read one physical line at a time. A physical line ends with a newline or linefeed character.
- In the MEMPTR version of the IMPORT statement, the MEMPTR must be pre-allocated to the size needed for reading. See the example, r-impmem.p, above.
- When importing a LONGCHAR variable, Progress uses the code page information in the exported file header to determine the variable's code page.

See also DEFINE STREAM statement, DISABLE TRIGGERS statement, DISPLAY statement, EXPORT statement, INPUT FROM statement, INPUT CLOSE statement, PUT statement, STRING function

INDEX function

Returns an integer that indicates the position of the target string within the source string.

Syntax

```
INDEX ( source , target [ , starting ] )
```

source

A CHARACTER or LONGCHAR expression.

target

A CHARACTER or LONGCHAR expression whose position you want to locate in *source*. If *target* does not exist within *source*, INDEX returns a 0.

starting

An integer that specifies at which left-most position in the string to start the search. For example, INDEX("abcdefabcdef","abc",6) returns 7.

Examples

For this example, you must enter 1, 2, 3, 4, or 5. The INDEX function checks if the digit exists in the string "12345".

r-index.p

```
DEFINE VARIABLE x AS CHARACTER FORMAT "9"
    LABEL "Enter a digit between 1 and 5".
DEFINE VARIABLE show AS CHARACTER FORMAT "x(5)" EXTENT 5 LABEL "Literal"
    INITIAL["One", "Two", "Three", "Four", "Five"].

REPEAT:
    SET x AUTO-RETURN.
    IF INDEX("12345",x) = 0 THEN DO:
        MESSAGE "Digit must be 1,2,3,4, or 5. Try again.".
        UNDO, RETRY.
    END.
    ELSE DISPLAY show[INTEGER(x)].
END.
```

This procedure also uses the *starting* option:

r-index2.p

```
DEFINE VARIABLE positions AS CHARACTER FORMAT "x(60)".
DEFINE VARIABLE sentence AS CHARACTER FORMAT "x(72)".
DEFINE VARIABLE vowel AS CHARACTER FORMAT "x".
DEFINE VARIABLE found AS INTEGER.
DEFINE VARIABLE loop  AS INTEGER.
DEFINE VARIABLE start AS INTEGER.

FORM sentence LABEL "Type in a sentence"
  WITH FRAME top
  TITLE "This program will tell where the vowels are in a sentence.".

SET sentence WITH FRAME top.
DO loop = 1 TO 5:
  positions = "".
  vowel = SUBSTRING("aeiou",loop,1).
  start = 1.
  found = INDEX(sentence,vowel,start).
  DO WHILE found > 0:
    positions = positions + STRING(found) + " ".
    start = found + 1.
    found = INDEX(sentence,vowel,start).
  END.
  DISPLAY
  vowel LABEL "Vowel"
  positions LABEL "Is found at locations..." WITH 5 DOWN.
  DOWN.
END.
```

Notes

- If either operand is case sensitive, then the search is case sensitive.
- If the target string is null, the result is 0.
- The INDEX function is double-byte enabled. You can specify *target* and *source* strings for the INDEX function that contain double-byte characters.

See also LOOKUP function, R-INDEX function

INPUT function

References the value of a field in a frame. For example, if you use the PROMPT-FOR statement to get input from the user, PROMPT-FOR stores that information in the screen buffer. You can use the INPUT function to refer to that information.

Note: Does not apply to SpeedScript programming.

Syntax

```
INPUT [ FRAME frame ] field
```

FRAME *frame*

The name of the frame that contains the field named by the *field* argument. If you do not name a frame, the INPUT function starts with the current frame and searches outward until it finds the field you name with the *field* argument.

field

The name of a field or variable whose value is stored in the screen buffer. The specified field must be viewed as a fill-in or text widget.

Example

This procedure displays the current credit-limit for a customer. The PROMPT-FOR statement prompts the user for a new credit-limit value and stores the supplied data in the screen buffer. The procedure uses the INPUT function to point to the data in that buffer.

r-input.p

```
FOR EACH CUSTOMER:
  DISPLAY cust-num name credit-limit LABEL "Current credit limit"
    WITH FRAME a 1 DOWN ROW 1.
  PROMPT-FOR credit-limit LABEL "New credit limit" WITH SIDE-LABELS
    NO-BOX ROW 10 FRAME b.
  IF INPUT FRAME b credit-limit <> credit-limit
  THEN DO:
    DISPLAY "Changing max credit of" name SKIP
            "from" credit-limit "to" INPUT FRAME b credit-limit
            WITH FRAME c ROW 15 NO-LABELS.
    credit-limit = INPUT FRAME b credit-limit.
  END.
  ELSE DISPLAY "No change in credit limit" WITH FRAME d ROW 15.
END.
```

If the user enters a new value, the procedure displays a message that the value has been changed. If the user enters the same value, the procedure displays a message that the credit-limit has not been changed.

Notes

- If you use a field or variable that is referenced with INPUT in more than one frame, then Progress uses the value in the frame most recently introduced in the procedure. To ensure that you are using the appropriate frame, use the FRAME option with the INPUT function to reference a particular frame.
- If you use the INPUT function for a character field whose format contains fill characters, then the value of the function does not contain the fill characters. The fill characters are not stored in the database field or variable, but are instead supplied during display formatting of the data.

INPUT CLEAR statement

Clears any keystrokes buffered from the keyboard, discarding any type-ahead characters. The INPUT CLEAR statement is useful when you want to make sure Progress clears out extra characters in the input statement that could follow a field entry that is too long.

Note: Does not apply to SpeedScript programming.

Syntax

```
INPUT CLEAR
```

Example

This menu procedure tests each key the user presses. If the user presses a key other than 1, 2, or 3, Progress clears the keyboard buffer and displays a message.

r-inclr.p

```
DISPLAY "     Please choose     " SKIP
  " 1  Run order entry    " SKIP
  " 2  Run receivables    " SKIP
  " 3  Exit               " WITH CENTERED FRAME menu.
REPEAT:
  READKEY.
  IF LASTKEY = KEYCODE("1") THEN RUN ordentry.
  ELSE
  IF LASTKEY = KEYCODE("2") THEN RUN receive.
  ELSE
  IF LASTKEY = KEYCODE("3") THEN QUIT.
  ELSE DO:
    MESSAGE "Sorry, that is not a valid choice".
    INPUT CLEAR.
  END.
END.
```

Notes

- In Windows, the keyboard type-ahead buffer can contain a maximum of 16 characters.
- If the current input source is not the keyboard, the INPUT CLEAR statement has no effect.
- INPUT CLEAR is not available on the Windows GUI platform after Progress Version 7.3D. It is available in Windows character and non-Windows GUI or character platforms.

See also

EDITING phrase

INPUT CLOSE statement

Closes the default input source or the stream you name.

Syntax

```
INPUT [ STREAM stream ] CLOSE
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry and *OpenEdge Development: Programming Interfaces* for more information on streams.

Example

Instead of getting input from the terminal, the following procedure gets input from a file named `r-in.dat`. The SEARCH function determines the full pathname of this file.

r-in.p

```
INPUT FROM VALUE(SEARCH("r-in.dat")).

REPEAT:
    PROMPT-FOR cust.cust-num credit-limit.
    FIND customer USING INPUT cust-num.
    ASSIGN credit-limit.
END.

INPUT CLOSE.
```

Here is what the contents of the `r-in.dat` file looks like:

```
1 55800
2 41300
5 88000
```

The PROMPT-FOR statement uses the first data item (1) as the cust-num and the second data item (55800) as the credit-limit. The FIND statement finds the customer whose cust-num is 1 and assigns the value of 55800 as that customer's credit limit. On the next iteration of the REPEAT block, the PROMPT-FOR statement uses the value of 2 as the cust-num, the value of 41300 as the credit-limit, etc.

The INPUT CLOSE statement closes the input source, resetting it to the terminal. When you run this procedure, the data in the window is simply an echo of the data as the procedure is reading it in from the `taxno.dat` file. If you do not want to display the data, add the word NO-ECHO to the end of the INPUT FROM statement.

Notes

- The default input source is the terminal unless the procedure was called by another procedure. In that case, the default input source is the one that was active in the calling procedure when the second procedure was called.
- When a procedure ends, Progress closes all input sources established in that procedure.
- For more information on input sources, see *OpenEdge Development: Programming Interfaces*.

See also DEFINE STREAM statement, INPUT FROM statement

INPUT FROM statement

Specifies a new input source.

Syntax

```
INPUT [ STREAM stream ] FROM
  {     opsys-file
      | opsys-device
      | TERMINAL
      | VALUE ( expression )
      | OS-DIR ( directory ) [ NO-ATTR-LIST ]
  }
  [ LOB-DIR { constant | VALUE ( expression ) } ]
  [ BINARY ]
  [ ECHO | NO-ECHO ]
  [ MAP protermcap-entry | NO-MAP ]
  [ UNBUFFERED ]
  [     NO-CONVERT
      | { CONVERT
            [ TARGET target-codepage ]
            [ SOURCE source-codepage ]
        }
  ]
```

`STREAM stream`

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry and *OpenEdge Development: Programming Interfaces* for more information on streams.

opsys-file

The name of the file that contains the data you want to input to a procedure. Remember that UNIX file names are case sensitive.

opsys-device

The name of a UNIX or Windows device.

`TERMINAL`

Indicates that you want to get input from your terminal. The terminal is the default input source. You cannot use TERMINAL with STREAM.

`VALUE ( expression )`

An expression whose value is the source where you want to input data.

`OS-DIR (directory) [ NO-ATTR-LIST ]`

Indicates that you want your input to be the filenames found in *directory*. The value of *directory* is a character expression specifying an operating system directory. If *directory* is not a directory or you do not have permission to read it, then the INPUT statement raises ERROR. Otherwise, Progress generates the directory list and feeds it back to the calling program through the INPUT stream. An INPUT CLOSE statement discards any unread filenames from the list.

Each line read from the input stream is a string composed of three tokens: the file's base name, the file's absolute path name, and an attribute list indicating the file type which consists of one or more of the characters listed below.

You will get one of the following characters:

- **F** — Regular file or FIFO pipe
- **D** — Directory
- **S** — Special device
- **X** — Unknown file type

You might also get one or more of the following characters:

- **H** — Hidden file
- **L** — Symbolic link
- **P** — Pipe file

If you specify the NO-ATTR-LIST option, you will not get the attribute list for any line read from the input stream.

The two filenames in each input line are in EXPORT format; that is, they are enclosed in quotes and any embedded quotes are doubled. This means that INPUT FROM can process any filename, containing any characters, as long as IMPORT is used to read the input.

NO-ATTR-LIST

Omits the attribute list indicating the file type. This can speed up program execution. An example of this form of the statement is as follows:

```
INPUT FROM OS-DIR("c:\mydir") NO-ATTR-LIST.
```

LOB-DIR { *constant* | VALUE (*expression*) }

Specifies the directory from which the IMPORT statement reads large object data files (such as BLOB and CLOB data files). The *constant* and *expression* arguments are character expressions that evaluate to an absolute pathname or a relative pathname (relative to the directory specified in *opsys-file*).

If the specified character expression evaluates to either the Unknown value (?) or a directory that does not exist, or you do not have permission to read the specified directory, Progress raises the ERROR condition.

The LOB-DIR option is valid only when you specify an operating system file as the input data source.

BINARY

Allows all input to be read directly without any conversion or interpretation. By default, NUL (\0) terminates character strings, and other control characters are interpreted as expected for the operating system.

ECHO

Displays all input data on the current output device. Data is echoed by default.

NO-ECHO

Accepts input data without displaying it on the current output device. If you do not use this option, INPUT FROM automatically displays input data on the current output device.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* value is an entry from the PROTERMCAP file. Use MAP to read from an input stream that uses a different character translation from the current stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). Progress uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make Progress bypass character translation altogether. See *OpenEdge Development: Internationalizing Applications* for more information on PROTERMCAP and national language support.

UNBUFFERED

Reads one character at a time from a normally buffered data source, such as a file. Use the UNBUFFERED option only when you can intermingle the input operations of a UNIX process, invoked with the Progress UNIX statement, with the input that follows the Progress INPUT FROM statement.

CONVERT

Allows you to modify the character conversions occurring between the external file and Progress. By default, the INPUT FROM statement converts characters from the code page specified with the Stream Code Page (-cpstream) parameter to the code page specified with the Internal Code Page (-cpinternal) parameter. If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name of the external file (instead of -cpstream). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the internal code page (instead of -cpinternal). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpstream to -cpinternal).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing `-cpinternal`). The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file (a binary file that contains all of the tables that Progress uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing `-cpstream`). The name that you specify must be a valid code page name available in the `DLC/convmap.cp` file (a binary file that contains all of the tables that Progress uses for character management).

`NO-CONVERT`

Specifies that no character conversions occur between the external file and memory. By default, the INPUT FROM statement converts characters from the -cpstream code page to the -cpinternal code page.

Example

Instead of getting input from the terminal, this procedure gets input from a file named `r-in.dat`. The SEARCH function determines the full pathname of this file.

r-in.p

```
INPUT FROM VALUE(SEARCH("r-in.dat")).

REPEAT:
    PROMPT-FOR cust.cust-num credit-limit.
    FIND customer USING INPUT cust-num.
    ASSIGN credit-limit.
END.

INPUT CLOSE.
```

This is what the contents of the `r-in.dat` file look like:

```
1 55800
2 41300
5 88000
```

The PROMPT-FOR statement uses the first data item (1) as the cust-num and the second data item (55800) as the credit-limit. The FIND statement finds the customer whose cust-num is 1 and assigns the value of 55800 as that customer's credit limit. On the next iteration of the REPEAT block, the PROMPT-FOR statement uses the value of 2 as the cust-num, the value of 41300 as the credit-limit, etc.

The INPUT CLOSE statement closes the input source, resetting it to the terminal. When you run this procedure, the data in the window is simply an echo of the data that the procedure is reading from the `taxno.dat` file. If you do not want to display the data, add the word NO-ECHO to the end of the INPUT FROM statement.

Notes

- To close the current input to a procedure, use the INPUT CLOSE statement. (The input source is automatically closed at the end of the procedure or when another default input source is opened.)
- The BINARY option allows you to use the READKEY statement to read control characters from the input source without interpretation. For example, NUL (\0) does not terminate strings, **CTRL+Z** does not signal EOF, and **CTRL+J** is not converted to **CTRL+M**, but their binary values are provided directly.
- If the input source and output destination are both the TERMINAL, then ECHO is always in effect.
- Use the INSERT, PROMPT-FOR, SET, or UPDATE statements to read data into a Progress procedure. The data is placed into the frame fields referenced in these statements, and, if you use ECHO, then the frame is output to the current output destination. If you use the NO-ECHO option, then the frame is not output. If a subsequent DISPLAY statement causes the frame to appear, then the input data also appears if the frame is not yet in view.
- SEEK is not supported in conjunction with the OS-DIR option.
- When using the OS-DIR option, the UNBUFFERED option is ignored. OS-DIR always buffers exactly one filename at a time.
- If you use the PROMPT-FOR, SET, or UPDATE statement to read data from a file, the FORMAT for the data is ignored. Therefore, if you rely on FORMAT to validate input, you might read invalid characters.
- If you use the PROMPT-FOR, SET, or UPDATE statement to read data from a file, and there is a piece of data in each line of the file that you want to disregard, use a caret (^) in the PROMPT-FOR, SET, or UPDATE statement. For more information on this symbol, see the reference entry for any of those statements.
- If end of file is reached, Progress responds as if you pressed **ENDKEY**.
- If a line consisting of a single period is read, that is treated as if you pressed **END-ERROR**. If the period is in quotes (".") it is treated as an ordinary character.

- When you use the INPUT FROM statement to read data from a file, there are two special characters you can use in that data file: tilde (~) and (slash (\) on UNIX, and hyphen (-).

  If characters in an input file take up more than one physical line, you can use tilde (~) to indicate a line continuation. This is an input file that uses a tilde:

```
92 "Match Point Tennis" "66 Homer Ave" "Como" ~
"TX" 75431
93 "Off the Wall" "20 Leedsville Ave" "Export" "PA" 15632
```

  Do not include a space after the tilde. For example:

Cust-num	Name	Address	City	State	Code
92	Match Point Tennis	66 Homer Ave	Como	TX	75431
93	Off The Wall	20 Leedsville Ave	Export	PA	15632

- You can see that the record containing the tilde was treated as a single input line.
- A hyphen in an input file indicates that you do not want to change the corresponding field in the INSERT, PROMPT-FOR, SET or UPDATE statement. This is the same input file as shown above, including the hyphen:

```
92 "Match Point Tennis" - "Como" "TX" 75431
93 "Off the Wall" "20 Leedsville Ave" "Export" "PA" 15632
```

  The procedure in the following example uses this file to set records in the customer file. When those records are displayed, the Match Point Tennis address does not change.

Cust-num	Name	Address	City	State	Code
92	Match Point Tennis		Como	TX	75431
93	Off The Wall	20 Leedsville Ave	Export	PA	15632

  To enter a literal hyphen from a file, enclose it in quotes ("-").

- In Windows, the data in the input file must have the following characteristics:
  - The lines of data in the file are separated by CR-LF pairs.
  - There is no **CTRL+Z** (EOF) embedded in the file.
- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that Progress uses for character management).
- If you specify a value of "undefined" for either *`source-codepage`* or *`target-codepage`*, no character conversion is performed.
- If the field being input is MEMPTR, you must use the BINARY and NO-CONVERT mode of operation to prevent your data from becoming corrupted if it contains binary data.
- With the BINARY and NO-CONVERT options, you will not get a translation of new-lines to the appropriate characters for your operating system and there will be no code page conversion between -cpinternal and -cpstream.
- If the field being input is MEMPTR and your MEMPTR contains ASCII data you may want code page conversion. However, you cannot get conversion by using the CONVERT parameter on the MEMPTR. You can get code page conversion by using the MEMPTR with the GET-STRING and CODEPAGE-CONVERT functions and the PUT-STRING statement.

See also DEFINE STREAM statement, INPUT CLOSE statement, INPUT THROUGH statement

INPUT THROUGH statement (NT, UNIX only)

Uses the output from a program as the input to a Progress procedure.

Syntax

```
INPUT [ STREAM stream ] THROUGH
  { program-name | VALUE ( expression ) }
  [ argument | VALUE ( expression ) ] ...
  [ ECHO | NO-ECHO ]
  [ MAP protermcap-entry | NO-MAP ]
  [ UNBUFFERED ]
  [     NO-CONVERT
    |  { CONVERT
           [ TARGET target-codepage ]
           [ SOURCE source-codepage ]
       }
  ]
```

`STREAM stream`

Specifies the name of a stream. If you do not name a stream, the unnamed stream is used. See the DEFINE STREAM statement reference entry and *OpenEdge Development: Programming Interfaces* for more information on streams.

`program-name`

Represents the name of the UNIX program where you are supplying data to a Progress procedure. This can be a standard UNIX command or your own program.

`VALUE ( expression)`

Specifies an expression whose value is the name of a UNIX program where you are supplying data to a Progress procedure.

Or, it is an expression whose value is an argument you want to pass to the UNIX program. INPUT THROUGH passes the value of *`expression`* as a character string.

argument

Represents an argument you want to pass to the UNIX program. INPUT THROUGH passes this *argument* as a character string.

If the *argument* is the literal value echo, no-echo, or unbuffered, enclose it in quotes to prevent Progress from interpreting that argument as one of the ECHO, NO-ECHO, or UNBUFFERED options for the INPUT THROUGH statement.

ECHO

Displays all input data on the current output destination. Data is echoed by default.

NO-ECHO

Accepts input data without displaying it on the current output device.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* value is an entry from the PROTERMCAP file. Use MAP to read an input stream that uses a different character translation from the current stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). Progress uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make Progress bypass character translation altogether. See *OpenEdge Deployment: Managing 4GL Applications* for more information on PROTERMCAP. See *OpenEdge Development: Internationalizing Applications* for more information on national language support.

UNBUFFERED

Reads one character at a time from a normally buffered data source, such as a file. Use the UNBUFFERED option only when the input operations of a UNIX process invoked by the Progress UNIX statement might be intermingled with the input from the Progress statements that follow the INPUT THROUGH statement.

CONVERT

Allows you to modify the character conversions occurring between the UNIX program and Progress. By default, the INPUT THROUGH statement converts characters from the code page specified with the Stream Code Page (-cpstream) parameter to the code page specified with the Internal Code Page (-cpinternal) parameter. If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name of the UNIX program (instead of -cpstream). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the internal code page (instead of -cpinternal). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpstream to -cpinternal).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the UNIX program and Progress. By default, the INPUT THROUGH statement converts characters from the -cpstream code page to the -cpinternal code page.

Examples

This procedure uses as its input source the output of the UNIX `echo` command. Before the command runs, the UNIX shell substitutes the process-id number for $$ and the current directory search path for $PATH. The results are then echoed and become available as a line of input to Progress. When the IMPORT statement is executed, the line of input from echo is read and the values are assigned to the two variables. Those variables can then be used for any purpose. In this example, the word `echo` must be lowercase and the word `$PATH` must be uppercase, since they both pass to UNIX:

r-ithru.p

```
DEFINE VARIABLE process-id AS CHARACTER.
DEFINE VARIABLE dir-path AS CHARACTER VIEW-AS EDITOR SIZE 60 BY 10.

INPUT THROUGH echo $$ $PATH NO-ECHO.

SET process-id dir-path WITH FRAME indata NO-BOX NO-LABELS.
DISPLAY process-id dir-path FORMAT "x(70)".

INPUT CLOSE.
```

When you use INPUT THROUGH, the UNIX program you name is executed as a separate process under its own shell. Therefore, the values of shell variables (such as $$) are values from that shell rather than the shell from which Progress executes.

The following procedure uses INPUT THROUGH twice to get input from the UNIX `pwd` and `ls` commands. The `pwd` command supplies the name of the current directory and the `ls` command supplies the name of each UNIX file in your current directory. After the variable `fn` is set, it displays on the screen.

r-ithru2.p

```
DEFINE VARIABLE dir-name AS CHARACTER FORMAT "x(64)".
DEFINE VARIABLE fn    AS CHARACTER FORMAT "x(32)".

FORM
  fn
  WITH DOWN FRAME dir-list.

/* Get the name of the current directory. */
INPUT THROUGH pwd NO-ECHO.
  SET dir-name.
INPUT CLOSE.

/* Use the directory name as a label for fn. */
fn:LABEL IN FRAME dir-list = dir-name.

/* List the directory. */
INPUT THROUGH ls NO-ECHO.

/* For each entry in the directory, read
   the entry into fn and display it.      */
REPEAT:
  SET fn WITH NO-BOX NO-LABELS FRAME indata.
  DISPLAY fn VIEW-AS TEXT WITH FRAME dir-list.
  DOWN WITH FRAME dir-list.
END.

INPUT CLOSE.
```

Notes

- INPUT THROUGH specifies the source for subsequent statements that process input. It does not read any data from the source.
- To use the IMPORT, INSERT, PROMPT-FOR, SET, or UPDATE statement, Progress puts the data in the frame fields referenced in these statements, and if ECHO is in effect, the frame is output to the current output destination. If you use the NO-ECHO option, then the frame is not output. If a subsequent DISPLAY statement causes the frame to display, the input data also displays.
- When INPUT THROUGH is closed, the pipe to the UNIX process is also closed.
- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that Progress uses for character management).
- If you specify a value of "undefined" for either *`source-codepage`* or *`target-codepage`*, no character conversion is performed.

See also DEFINE STREAM statement, INPUT CLOSE statement, INPUT FROM statement

INPUT-OUTPUT CLOSE statement (NT, UNIX only)

Closes a specified or default stream opened by an INPUT-OUTPUT THROUGH statement.

Syntax

```
INPUT-OUTPUT [ STREAM stream ] CLOSE
```

STREAM *stream*

The name of the stream you want to close. If you do not name a stream, Progress closes the default stream used by an INPUT-OUTPUT THROUGH statement.

Example

This procedure uses a C program to recalculate the price of each item in inventory. Specifically, the C program increases the price of each item by 3% or by 50 cents, whichever is greater. The INPUT-OUTPUT THROUGH statement tells the procedure to get its input from, and send its output to, the `r-iothru.p` procedure. The INPUT-OUTPUT CLOSE statement resets the input source to the terminal and the output destination to the terminal.

r-iothru.p

```
FOR EACH item WHERE item-num < 10:
    DISPLAY item-num price LABEL "Price before recalculation".
END.

INPUT-OUTPUT THROUGH r-iothru UNBUFFERED.

FOR EACH item WHERE item-num < 10:
    EXPORT price.
    SET price.
END.

INPUT-OUTPUT CLOSE.

FOR EACH item WHERE item-num < 10 WITH COLUMN 40:
    DISPLAY item-num price LABEL "Price after recalculation".
END.
```

Note

For more information, see *OpenEdge Development: Programming Interfaces*.

See also

DEFINE STREAM statement, INPUT-OUTPUT THROUGH statement

INPUT-OUTPUT THROUGH statement (NT, UNIX only)

Names a program (process) for Progress to start. This process is the input source as well as the output destination for the procedure.

Syntax

```
INPUT-OUTPUT [ STREAM stream ]
  THROUGH { program-name | VALUE ( expression ) }
  [ argument | VALUE ( expression ) ] ...
  [ ECHO | NO-ECHO ]
  [ MAP protermcap-entry | NO-MAP ]
  [ UNBUFFERED ]
  [     NO-CONVERT
     |  { CONVERT
             [ TARGET target-codepage ]
             [ SOURCE source-codepage ]
        }
  ]
```

`STREAM stream`

Specifies the name of a stream. If you do not name a stream, the unnamed stream is used. See the DEFINE STREAM statement reference entry and *OpenEdge Development: Programming Interfaces* for more information on streams.

program-name

Identifies the name of the UNIX program where the procedure is getting data and where the procedure is sending data.

`VALUE ( expression )`

Represents an expression whose value is the name of a UNIX program where the procedure is getting data and where the procedure is sending data.

Or, it is an expression whose value is an argument you want to pass to the UNIX program. INPUT-OUTPUT THROUGH passes the value of *expression* as a character string.

argument

Specifies an argument you want to pass to the UNIX program. INPUT-OUTPUT THROUGH passes this argument as a character string.

If the argument is the literal value echo, no-echo, or unbuffered, you must enclose it in quotes to prevent Progress from interpreting that argument as one of the ECHO, NO-ECHO, or UNBUFFERED options for the INPUT-OUTPUT THROUGH statement.

ECHO

Displays all input data to the unnamed stream. Data is not echoed by default.

NO-ECHO

Accepts input data without displaying it on the current unnamed stream. Data is not echoed by default.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* value is an entry from the PROTERMCAP file. MAP allows you to send output to and receive input from an I/O stream that uses different character translation than the current stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). Progress uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make Progress bypass character translation altogether. See *OpenEdge Deployment: Managing 4GL Applications* for more information on PROTERMCAP. See *OpenEdge Development: Internationalizing Applications* for more information on national language support.

UNBUFFERED

Reads and writes one character at a time from a normally buffered data source, such as a file. Use the UNBUFFERED option only when the input-output operations of a process invoked by Progress's UNIX statement can be intermingled with the input-output from the Progress statements that follow the INPUT-OUTPUT THROUGH statement. INPUT-OUTPUT THROUGH handles the buffering of data between the Progress procedure and the UNIX program that it invokes. Use the UNBUFFERED option if your procedure invokes any other programs with the UNIX statement.

CONVERT

Allows you to modify the character conversions occurring between the UNIX program and Progress. By default, the INPUT-OUTPUT THROUGH statement converts characters from the Stream Code Page (-cpstream) parameter to the code page specified with the Internal Code Page (-cpinternal) parameter as data received from *program-name*. As data is passed to *program-name*, then INPUT-OUTPUT THROUGH converts from the -cpinternal to -cpstream. If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name of the UNIX program (instead of -cpstream). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the internal code page (instead of -cpinternal). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpstream to -cpinternal).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the UNIX program and Progress. By default, the INPUT-OUTPUT THROUGH statement converts characters from the -cpstream code page to the -cpinternal code page as data is received from *program-name*. As data is passed to *program-name*, then INPUT-OUTPUT THROUGH converts from the -cpinternal to -cpstream.

Examples

This procedure uses a C program to recalculate the price of each item in inventory. Specifically, the C program increases the price of each item by 3% or by 50 cents, whichever is greater. The INPUT-OUTPUT THROUGH statement tells the procedure to get its input from, and send its output to, the `r-iothru.p` procedure. The INPUT-OUTPUT CLOSE statement resets the input source to the terminal and the output destination to the terminal.

r-iothru.p

```
FOR EACH item WHERE item-num < 10:
    DISPLAY item-num price LABEL "Price before recalculation".
END.

INPUT-OUTPUT THROUGH r-iothru UNBUFFERED.

FOR EACH item WHERE item-num < 10:
    EXPORT price.
    SET price.
END.

INPUT-OUTPUT CLOSE.

FOR EACH item WHERE item-num < 10 WITH COLUMN 40:
    DISPLAY item-num price LABEL "Price after recalculation".
END.
```

You can perform this calculation within a single Progress procedure. The C program is used for illustration purposes only. Use a UNIX program outside Progress to execute specialized calculations or processing.

You must unpack the C program from the `proguide` subdirectory and compile it before you can use it with the `r-iothru.p` procedure. If you do not have a C compiler, do not try this example.

This is the C program used by the `r-iothru.p` procedure:

r-iothru.c

```
#include <stdio.h>
#define MAX(a,b)    ( (a < b) ? b : a )

main( )
{
  float cost;

  /* This is important so that buffering does not   */
  /* defeat attempts to actually write on the pipe. */
  setbuf(stdout, (char *) NULL);

  while (scanf("%f", &cost) == 1) {
    /* Here the item cost is manipulated. We are */
    /* increasing the cost by a fixed percentage */
    /* (with a minimum increase), to provide     */
    /* an example.                               */
    cost = cost + MAX( 0.03 * cost, .50);
    printf("%10.2f\n", cost);
  }
}
```

Notes

- Use EXPORT or PUT, not DISPLAY, to write data to the program.
- Use SET to read data from the program.
- If you read data from a C program, put an upper limit on how many errors can occur before the program ends. Also remember that if the program prints an error message, that message is sent to Progress as data. You can use fprintf(stderr,...) to display debugging messages to the window, even in the middle of an INPUT-OUTPUT THROUGH operation.
- With INPUT-OUTPUT THROUGH in non-interactive mode, a Progress procedure can send information to a UNIX program, and the program can process that information and send the results back to Progress. Some UNIX utilities you can use in batch mode are `wc` (word count) and `sort`.

Here are some pointers for using INPUT-OUTPUT THROUGH in this way:

- When the procedure finishes sending data to the program, use the OUTPUT CLOSE statement to reset the standard output stream to the screen. Doing this signals an EOF on the pipe, indicating that the program has received all input. When the procedure has received all data from the program, use the INPUT CLOSE statement to reset the standard input stream. Do not use the INPUT-OUTPUT CLOSE statement, because that closes both pipes at once.
- If you want to use the INPUT-OUTPUT THROUGH statement with a UNIX utility that buffers its output, use the non-interactive approach.
- To signal an EOF, use OUTPUT CLOSE (rather than attempting to send a **CTRL+D**).

When you use INPUT-OUTPUT THROUGH in interactive mode Progress sends data to the program, and the program sends data back to Progress, etc.

Here are some pointers for using INPUT-OUTPUT THROUGH in this way:

- At the end of the interaction between the procedure and the program, use the INPUT-OUTPUT CLOSE statement to shut down both pipes.
- Be sure that the program you are using does not buffer its output. If the program is a C program, the first line of the program should be “setbuf(stdout, (char *) NULL):”. The program should also include “#include <stdio.h>”. These tell UNIX that the standard output of the program is unbuffered. If the program does buffer its output, use the batch approach to INPUT-OUTPUT THROUGH as explained in the previous note.
- If the program ends on some condition other than detecting an EOF, make sure that it tells the Progress procedure that it is about to end.

- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that Progress uses for character management).
- If you specify a value of “undefined” for either *`source-codepage`* or *`target-codepage`*, no character conversion is performed.

See also DEFINE STREAM statement, INPUT CLOSE statement, INPUT-OUTPUT CLOSE statement

INSERT statement

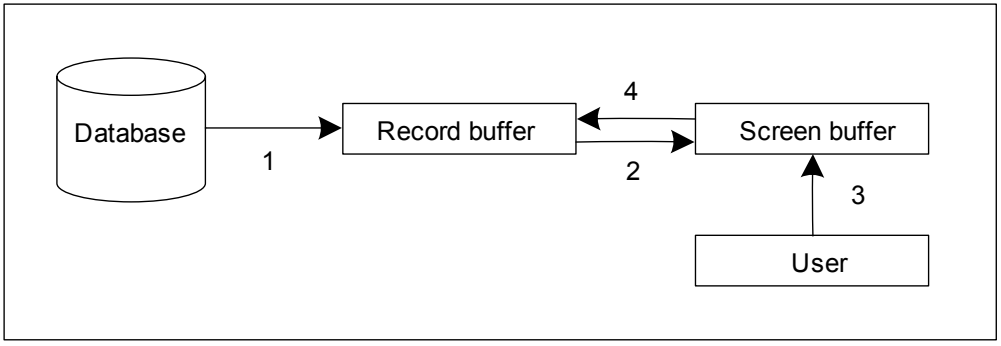
Creates a new database record, displays the initial values for the fields in the record, prompts for values of those fields, and assigns those values to the record.

The INSERT statement is a combination of the following statements:

- **CREATE** — Creates an empty record buffer.
- **DISPLAY** — Moves the record from the record buffer into the screen buffer and displays the contents of the buffer on the screen.
- **PROMPT-FOR** — Accepts input from the user, and puts that input into the screen buffer.
- **ASSIGN** — Moves data from the screen buffer into the record buffer.

Note: Does not apply to SpeedScript programming.

Data movement



1. **CREATE** — Creates an empty record buffer.
2. **DISPLAY** — Moves the contents of the record buffer to the screen buffer and displays the screen buffer.
3. **PROMPT-FOR** — Accepts input from the user into the screen buffer.
4. **ASSIGN** — Moves the contents of the screen buffer to the record buffer.

Syntax

```
INSERT record [ EXCEPT field ... ]
  [ USING { ROWID ( nrow ) | RECID ( nrec ) } ]
  [ frame-phrase ]
  [ NO-ERROR ]
```

record

The name of the record you want to add to a database file. Progress creates one record buffer for every file you use in a procedure. This buffer is used to hold a single record from the file associated with the buffer. Use the DEFINE BUFFER statement to create additional buffers, if necessary. The CREATE part of the INSERT statement creates an empty record buffer for the file in which you are inserting a record.

To insert a record in a file defined for multiple databases, you must qualify the record's filename with the database name. See the Record phrase reference entry for more information.

EXCEPT *field*

Inserts all fields except those listed in the EXCEPT phrase.

USING { ROWID (*nrow*) | RECID (*nrec*) }

Allows you to insert a record in an RMS relative file (for backward compatibility only) using a specific record number, where *nrow* is the ROWID relative record number of the record you want to insert and *nrec* is the RECID relative record number of the record you want to insert.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

NO-ERROR

Specifies that any errors that occur when you try to insert the record are suppressed. After the INSERT statement completes, you can check the ERROR-STATUS system handle for information on errors that have occurred.

Example

In this procedure the user adds a new order record. After the user adds a new order record, the procedure creates order-lines for that record. The procedure uses the CREATE statement to create order-lines rather than the INSERT statement. When you use the INSERT statement, the PROMPT-FOR and ASSIGN parts of the INSERT let you put data into all the fields of the record being inserted. In the case of order-lines, this procedure only lets you add information into a few of the order-line fields. Use CREATE together with UPDATE to single out the order-line fields.

r-insrt.p

```
REPEAT:
  INSERT order WITH 1 COLUMN.
  REPEAT:
    CREATE order-line.
    order-line.order-num = order.order-num.
    UPDATE line-num order-line.item-num qty price.
    /* Verify the item-num by finding an item
      with that number */
    FIND item OF order-line.
  END.
END.
```

Notes

- If an error occurs during the INSERT statement, Progress retries the data entry part of the statement and processes the error associated with the block that contains the statement. (For example, an error might occur when the user enters a duplicate index value for a unique index.)
- Any frame characteristics described by an INSERT statement contribute to the frame definition. When Progress compiles a procedure, it uses a top-to-bottom pass of the procedure to design all the frames required by that procedure, including those referenced by INSERT statements, and adds field and related format attributes as it goes through the procedure.
- If you receive input from a device other than the terminal, and the number of characters read by the INSERT statement for a particular field or variable exceeds the display format for that field or variable, Progress returns an error. However, if you set a logical field that has a format of y/n and the data file contains a value of yes or no, Progress converts that value to y or n.

- If you use a single qualified identifier with the INSERT statement, the compiler first interprets the reference as *dbname.filename*. If the compiler cannot resolve the reference as *dbname.filename*, it tries to resolve it as *filename.fieldname*.

  When inserting fields, you must use filenames that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.

- The INSERT statement causes any related database CREATE triggers to execute. All CREATE triggers execute after the record is actually created. If a CREATE trigger fails (or executes a RETURN statement with the ERROR option), the record creation is undone. See *OpenEdge Development: Progress 4GL Handbook* for more information on database triggers.

See also DEFINE BUFFER statement, Frame phrase

INTEGER function

Converts an expression of any data type to an integer value, rounding that value if necessary.

Syntax

```
INTEGER ( expression )
```

expression

A constant, field name, variable name, or expression whose value can be of any data type. If the value of *expression* is character then it must be valid for conversion into a number (for example, “1.67" is valid, “1.x3" is not). If *expression* is logical, then the result is 0, if *expression* is FALSE, and 1, if *expression* is TRUE. If *expression* is a date, then the result is the number of days from 1/1/4713 B.C. to that day. If the value of *expression* is the Unknown value (?), then the result is the Unknown value (?).

Example

This procedure takes the first word (that is, the substring that precedes the first space character) from the customer address and tries to convert it to an integer (street-number). If the conversion fails (for example, the first word contains non-numeric characters) the procedure displays an error message. Otherwise the cust-num, address, and converted street number are displayed.

r-intgr.p

```
DEFINE VARIABLE street-number AS INTEGER LABEL "Street Number".

FOR EACH customer:
   ASSIGN street-number = INTEGER(ENTRY(1, address, " ")) NO-ERROR.

   IF ERROR-STATUS:ERROR
   THEN MESSAGE "Could not get street number of" address.
   ELSE DISPLAY cust-num address street-number.
END.
```

See also

DECIMAL function, STRING function

INTERFACE statement

Defines an interface. An interface is a special type of class definition that contains a set of method prototype declarations for common methods implemented by one or more classes. All classes that implement the interface must support the methods declared in the interface.

Note: This statement is applicable only when used in a class definition (`.cls`) file.

Syntax

```
INTERFACE type-name:

interface-body
```

type-name

A character string that specifies the type name of an interface. Specify an interface type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

interface-body

The body of an interface definition is composed of the following types of elements:

- Temp-table or ProDataSet object definitions used as parameters by one or more methods whose prototype is declared in this interface.
- Method prototypes for common methods implemented by one or more classes.

Define elements in the interface body using the following syntax:

```
[ { temp-table | dataset } ... ]
[ method-prototypes ]
END [ INTERFACE ].
```

temp-table | *dataset*

Specifies one or more temp-table or ProDataSet object definitions used as parameters by one or more methods declared in this interface. You must specify these object definitions before any method prototypes. Progress does not allocate memory for these object definitions. You cannot specify an access mode for these object definitions.

The definition of temp-table and ProDataSet object parameters for methods defined in any classes that implement this interface must match the temp-table or ProDataSet object definitions in this interface.

For temp-table objects:

- The temp-tables must have the same number of fields, and each field must match with respect to the data type, extent, and position. Neither the table names nor field names must match.
- The temp-tables must have the same number of indexes, and each index component must match, including the index names. However, the index-component field names do not need to match.

For ProDataSet objects:

- The ProDataSet objects must have the same number of member buffers, and the buffers must be in the same order. Neither the buffer names nor ProDataSet names must match.
- The temp-tables of these buffers must match as described above.

method-prototypes

Specifies one or more method prototypes in the interface. A method prototype declares a method in the class without implementing the method (that is, without specifying the method's logic or the END METHOD statement). You must specify a method prototype with a PUBLIC access mode. For more information, see the METHOD statement reference entry in this book.

The implementation of these method prototypes, in other classes that implement this interface, must match these declarations with respect to the access mode of the method and its return type, and with respect to the number, data type, and access mode of any parameters.

```
END [ INTERFACE ]
```

Specifies the end of the interface body definition. You must end the interface body definition with the END statement.

Example The following example shows the definition of an interface that declares two method prototypes:

```
INTERFACE acme.myObjs.Interfaces.IBusObj:

  METHOD PUBLIC VOID printObj(INPUT deviceName AS CHARACTER).
  METHOD PUBLIC VOID logObj(INPUT deviceName AS CHARACTER).

END INTERFACE.
```

The following example shows the definition of a class that implements this interface and its method declarations:

```
CLASS acme.myObjs.CustObj INHERITS acme.myObjs.Common.CommonObj
  IMPLEMENTS acme.myObjs.Interfaces.IBusObj:
         .
         .
         .
/ * Implement the methods declared in the interface */

     METHOD PUBLIC VOID printObj(INPUT deviceName AS CHARACTER):
         MESSAGE "In CustObj class - printObj method" VIEW-AS ALERT-BOX.
     END METHOD.

     METHOD PUBLIC VOID logObj(INPUT deviceName AS CHARACTER):
         MESSAGE "In CustObj class - logObj method" VIEW-AS ALERT-BOX.
     END METHOD.

END CLASS.
```

Notes

- You can terminate an INTERFACE statement with either a period (.) or a colon (:).
- A complete interface definition must begin with the INTERFACE statement and end with the END statement.
- The access mode for an interface definition is always PUBLIC.
- A class definition (`.cls`) file can contain only one interface definition.
- The compiled version of an interface definition file is an r-code (`.r`) file.
- You can define an object reference variable for an interface, which lets you access the interface or a class that implements the interface, but you **cannot** create an instance of an interface with the NEW statement. Instead, create an instance of a class that implements the interface. For more information, see the AS CLASS option in the DEFINE VARIABLE statement reference entry in this book.
- You can reference include files from within an interface definition. For more information about include files, see the { } Include file reference entry in this book.
- All built-in preprocessor directives are supported in interface definitions.
- All built-in preprocessor names are supported in interface definitions. For a list of preprocessor name, see the { } Preprocessor name reference entry in this book.
- You cannot pass compile-time arguments to interface definition files. However, you can pass compile-time arguments to include files referenced in an interface definition file.
- You can store class definition r-code files in Progress procedure libraries. If Progress encounters a procedure library on `PROPATH`, it will search the library for the specified r-code. However, you cannot execute r-code files stored in a procedure library that is not on `PROPATH` using the *`procedure-library-path`*`<<`*`member-name`*`>>` syntax.

See also CLASS statement, METHOD statement

INTERVAL function

Returns the time interval between two DATE, DATETIME, or DATETIME-TZ values.

Syntax

```
INTERVAL (datetime1, datetime2, interval-unit)
```

datetime1

An expression whose value is a DATE, DATETIME, or DATETIME-TZ.

datetime2

An expression whose value is a DATE, DATETIME, or DATETIME-TZ.

interval-unit

A character constant, or a character expression that evaluates to one of the following time units: 'years', 'months', 'weeks', 'days', 'hours', 'minutes', 'seconds' or 'milliseconds'. These values are case insensitive and may be singular.

Notes

- This function returns a signed integer value (positive or negative). For example, if *datetime1* is less than *datetime2*, the INTERVAL function returns a negative value.
- If *datetime1* or *datetime2* is a DATE or DATETIME, the time value defaults to midnight and the time zone value defaults to the session's time zone, respectively.
- You are responsible for managing value overflow, if any.

IS-ATTR-SPACE function

This function is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
IS-ATTR-SPACE
```

Example

This procedure displays a message indicating whether the current terminal is space-taking:

r-isattr.p

```
DEFINE VARIABLE termtype AS LOGICAL FORMAT "spacetaking/non-spacetaking".

termtype = IS-ATTR-SPACE.

DISPLAY "You are currently using a" termtype NO-LABEL "terminal"
      WITH FRAME d1 CENTERED ROW 5.
```

Note

If you run Progress in batch mode, IS-ATTR-SPACE returns the Unknown value (?).

See also

TERMINAL statement

IS-CODEPAGE-FIXED( ) function

Returns TRUE if the code page of the specified LONGCHAR variable is fixed; otherwise it returns FALSE.

Syntax

```
IS-CODEPAGE-FIXED ( longchar )
```

longchar

The name of a LONGCHAR variable.

See also FIX-CODEPAGE function, GET-CODEPAGE function

IS-COLUMN-CODEPAGE( ) function

Returns TRUE if the specified CLOB field is a COLUMN-CODEPAGE CLOB. Otherwise, it returns FALSE (that is, if the CLOB is a DBCODEPAGE CLOB or a TTCODEPAGE CLOB).

Syntax

```
IS-COLUMN-CODEPAGE ( field )
```

field

The name of a CLOB field.

IS-LEAD-BYTE function

Returns YES if the first character of the string is the lead-byte of a multi-byte character. Returns NO if it is not.

Syntax

```
IS-LEAD-BYTE ( string )
```

string

A character expression (a constant, field name, variable name, or any combination of these) whose value is a character.

Example

In this example, IS-LEAD-BYTE returns YES because the first byte of the first character is the lead-byte of a double-byte character. The output is “Lead: yes”.

```
DEFINE VARIABLE Lead AS LOGICAL.
Lead = IS-LEAD-BYTE ("中 xy").
DISPLAY Lead WITH 1 COLUMN.
```

See also

OVERLAY statement, SUBSTRING statement

ISO-DATE function

Returns a character representation of a DATE, DATETIME , or DATETIME-TZ that conforms to the ISO 8601 standard for date/time representations.

Note: These formats are equivalent to the XML Schema `date` and `dateTime` formats.

Syntax

```
ISO-DATE ( expression  )
```

expression

An expression that evaluates to a DATE, DATETIME or DATETIME-TZ.

The ISO-DATE function returns the character string in the standard ISO format of the data type. Table 36 lists the standard ISO formats for each data type.

Table 36: Standard ISO formats

Data type	ISO format
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DDTHH:MM:SS.SSS
DATETIME-TZ	YYYY-MM-DDTHH:MM:SS.SSS+HH:MM

See also

DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, MONTH function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

KBLABEL function

Returns the keyboard label (such as **F1**) of the key that performs a specified Progress function (such as GO).

Note: Does not apply to SpeedScript programming.

Syntax

```
KBLABEL ( key-function )
```

key-function

An expression whose value is the name of the special Progress key function. See *OpenEdge Development: Programming Interfaces* for possible values of *key-name*. If *key-function* is a constant, enclose it in quotation marks (" "). See the same chapter for a list of key functions and the corresponding standard keyboard keys.

Example

The `r-kblabl.p` procedure allows the user to update some of the fields in each of the customer records. It also displays a message in the status message area at the bottom of the window.

r-kblabl.p

```
STATUS INPUT "Enter data, then press " + KBLABEL("GO").
FOR EACH customer:
  UPDATE cust-num name address city state.
END.
```

Note

If you reassign a new function key for the key function with the ON statement, the KBLABEL function returns the new key.

KEYCODE function

Evaluates a key label (such as **F1**) for a key in the predefined set of keyboard keys and returns the corresponding integer key code (such as 301). See *OpenEdge Development: Programming Interfaces* for a list of key codes and key labels.

Note: Does not apply to SpeedScript programming.

Syntax

```
KEYCODE ( key-label )
```

key-label

A constant, field name, variable name, or expression that evaluates to a character string that contains a key label. If *key-label* is a constant, enclose it in quotation marks (" ").

Example

This procedure displays a menu and highlights different selections on the menu depending on which key you press. On the first iteration of the REPEAT block, the COLOR statement tells Progress to color msg[i] with the same color used to display messages. Because the initial value of i is 1, msg[i] is the first menu selection. Therefore, the first menu selection is colored MESSAGES.

r-keycod.p

```
DEFINE VARIABLE msg  AS CHARACTER EXTENT 3.
DEFINE VARIABLE i    AS INTEGER INITIAL 1.
DEFINE VARIABLE newi AS INTEGER INITIAL 1.

DISPLAY "     Please choose     " SKIP(1)
  " 1  Run order entry    " @ msg[1]
    ATTR-SPACE SKIP
  " 2  Run receivables    " @ msg[2]
    ATTR-SPACE SKIP
  " 3  Exit               " @ msg[3]
    ATTR-SPACE SKIP
  WITH CENTERED FRAME menu NO-LABELS.

REPEAT:
    COLOR DISPLAY MESSAGES msg[i] WITH FRAME menu.
    READKEY.
    IF LASTKEY = KEYCODE("CURSOR-DOWN") AND i < 3
    THEN newi = i + 1.
    ELSE IF LASTKEY = KEYCODE("CURSOR-UP") AND i > 1
    THEN newi = i - 1.
    ELSE IF LASTKEY = KEYCODE("GO") OR
       LASTKEY = KEYCODE("RETURN")
    THEN LEAVE.

    IF i <> newi THEN COLOR DISPLAY NORMAL
      msg[i] WITH FRAME menu.
    i = newi.
END.
```

Here's what happens if you press the cursor-down key:

1. The READKEY statement reads the value of the key you pressed.
2. The first IF . . . THEN . . . ELSE statement tests to see if the key code of the key you pressed is CURSOR-DOWN. It also checks whether the value of i is less than 3. Both of these things are true, so the procedure adds one to the value of newi, making newi equal two.
3. The next two IF statements are ignored because the condition in the first IF statement was true. The procedure continues on the last IF statement: IF i <> newi THEN COLOR DISPLAY NORMAL msg[i] WITH FRAME menu.
4. Remember, i is still 1 but newi is now 2. Thus, i is not equal to newi. Which means that the IF statement test is true. Therefore, Progress colors msg[i], which is still msg[1] (the first menu selection), NORMAL. So the first menu selection is no longer highlighted.
5. Just before the end of the REPEAT block, i is set equal to newi. Which means that msg[i] is now msg[2], or the second menu selection.
6. On the next iteration, the COLOR statement colors msg[i], that is the second menu selection, MESSAGES. The end result of pressing **CURSOR-DOWN** is that the highlight bar moves to the second menu selection.

See also KEYFUNCTION function, KEYLABEL function

KEYFUNCTION function

Evaluates an integer expression (such as 301) and returns a character string that is the function of the key associated with that integer expression (such as GO).

Note: Does not apply to SpeedScript programming.

Syntax

```
KEYFUNCTION ( expression )
```

expression

A constant, field name, variable name, or expression whose value is an integer key code.

Example

This procedure displays a menu and highlights different selections, depending on which key you press. On the first iteration of the REPEAT block, the COLOR statement tells Progress to color msg[i] with the same color used to display messages. Because the initial value of i is 1, msg[i] is the first menu selection. Therefore, the first menu selection is colored MESSAGES.

r-keyfn.p

```
DEFINE VARIABLE msg  AS CHARACTER EXTENT 3.
DEFINE VARIABLE i    AS INTEGER INITIAL 1.
DEFINE VARIABLE newi AS INTEGER INITIAL 1.
DEFINE VARIABLE func AS CHARACTER.

DISPLAY "     Please choose     " SKIP(1)
   " 1  Run order entry    " @ msg[1] ATTR-SPACE SKIP
   " 2  Run receivables    " @ msg[2] ATTR-SPACE SKIP
   " 3  Exit               " @ msg[3] ATTR-SPACE SKIP
   WITH CENTERED FRAME menu NO-LABELS.

REPEAT:
  COLOR DISPLAY MESSAGES msg[i] WITH FRAME menu.
  READKEY.
  func = KEYFUNCTION(LASTKEY).
  IF func = "CURSOR-DOWN" AND i < 3
  THEN newi = i + 1.
  ELSE IF func = "CURSOR-UP" AND i > 1
  THEN newi = i - 1.
  ELSE IF func = "GO" OR func = "RETURN"
  THEN LEAVE.
  IF i <> newi THEN COLOR DISPLAY NORMAL
    msg[i] WITH FRAME menu.
  i = newi.
END.
```

See the example in the KEYCODE function reference entry for details on what happens if you press keylabel component.

Notes

- The value returned by the KEYFUNCTION function is affected by any ON statements you use to redefine the value of the key represented by *expression*.
- If the key represented by *expression* has no function currently assigned to it or if it has the function of BELL, KEYFUNCTION returns a null value.
- KEYFUNCTION(-2) is equal to ENDKEY.

See also

KEYCODE function, KEYLABEL function

KEYLABEL function

Evaluates a key code (such as 301) and returns a character string that is the predefined keyboard label for that key (such as **F1**).

Note: Does not apply to SpeedScript programming.

Syntax

```
KEYLABEL ( key-code )
```

key-code

The key code of the key whose label you want to know. A special case of *key-code* is LASTKEY. See *OpenEdge Development: Programming Interfaces* for a list of key codes and key labels.

Example

This procedure reads each keystroke the user makes, leaving the procedure only when the user presses **GO**. The KEYLABEL function tests the LASTKEY pressed, and returns the label of the key. (Remember that the value in LASTKEY is the key code of the last key pressed.)

r-keylbl.p

```
DISPLAY "Press the " + KBLABEL("GO") + " key to leave procedure"
     FORMAT "x(50)".
REPEAT:
      READKEY.
      HIDE MESSAGE.
      IF LASTKEY = KEYCODE(KBLABEL("GO")) THEN RETURN.
      MESSAGE "Sorry, you pressed the"  KEYLABEL(LASTKEY) "key.".
END.
```

Note

Some key codes can be associated with more than one key label. The KEYLABEL function always returns the label listed first in the Progress table of key labels.

See also

KEYCODE function, KEYFUNCTION function

KEYWORD function

Returns a character value that indicates whether a string is a Progress reserved keyword.

Syntax

```
KEYWORD ( expression )
```

expression

A constant, field name, variable name, or expression that results in a character string. If expression matches a Progress reserved keyword or valid abbreviation of a reserved keyword, the KEYWORD function returns the full keyword. If there is no match, the KEYWORD function returns the Unknown value (?).

In some cases, the abbreviation for a keyword is also a keyword. For example, if *expression* is "def" (the abbreviation for DEFINE) or "col" (the abbreviation for COLUMN), the KEYWORD function returns the values "def" and "col", respectively.

If you use Progress Run Time, the KEYWORD function always returns the Unknown value (?).

Example

In this example, the KEYWORD function tests the value of formname. If the user tries to use a reserved word as a form name, Progress displays a message to try again.

r-keywd.p

```
DEFINE VARIABLE formname AS CHARACTER FORMAT "x(20)".

REPEAT ON ERROR UNDO, RETRY:
  UPDATE formname.
  IF KEYWORD(formname) NE ?
  THEN DO:
    MESSAGE formname + " may not be used as a form name".
    UNDO, RETRY.
  END.
  ELSE LEAVE.
END.
```

Notes

- Because KEYWORD recognizes abbreviations, it does not distinguish between FORM and FORMAT or between ACCUM and ACCUMULATE.
- This function returns the Unknown value (?) for colors and most data types, as well as all unreserved keywords. See the "Keyword Index" section on page 2209 for a list of Progress reserved and unreserved keywords.
- KEYWORD is less restrictive than the KEYWORD-ALL function. Use this function if you do not want to use Progress reserved keywords as field names, for example.
- For SpeedScript, all Progress reserved keywords are also reserved for SpeedScript.

See also KEYWORD-ALL function

KEYWORD-ALL function

Returns a character value that indicates whether a string is a Progress keyword. This function returns all keywords and does not distinguish between reserved or unreserved keywords.

Syntax

```
KEYWORD-ALL ( expression )
```

expression

A constant, field name, variable name, or expression that results in a character string. If expression matches a Progress keyword, whether reserved or unreserved or valid abbreviation of a keyword, the KEYWORD-ALL function returns the full keyword. If there is no match, the KEYWORD-ALL function returns the Unknown value (?).

KEYWORD-ALL is the same function as KEYWORD in Progress Version 6 and earlier. Use this function if you do not want to use Progress reserved and unreserved keywords as field names, for example.

In some cases, the abbreviation for a keyword is also a keyword. For example, if *expression* is “def” (the abbreviation for DEFINE) or “col” (the abbreviation for COLUMN), the KEYWORD function returns the values “def” and “col”, respectively.

If you use Progress Run Time, the KEYWORD-ALL function always returns the Unknown value (?).

Example

In this example, the KEYWORD-ALL function tests the value of formname. If the user tries to use a keyword as a form name, Progress displays a message to try again.

r-keywda.p

```
DEFINE VARIABLE formname AS CHARACTER FORMAT "x(20)".

REPEAT ON ERROR UNDO, RETRY:
    UPDATE formname.
    IF KEYWORD-ALL(formname) NE ?
    THEN DO:
      MESSAGE formname + "cannot be used as a form name".
      UNDO, RETRY.
    END.
    ELSE LEAVE.
END.
```

Notes

- Because KEYWORD-ALL recognizes abbreviations, it does not distinguish between FORM and FORMAT or between ACCUM and ACCUMULATE.
- This function returns the Unknown value (?) for colors and most data types, as well as all unreserved keywords. See the "Keyword Index" section on page 2209 for a list of Progress reserved and unreserved keywords.
- For SpeedScript, all Progress reserved keywords are also reserved for SpeedScript.

See also KEYWORD function

LAST function

Returns a TRUE value if the current iteration of a DO, FOR EACH, or REPEAT . . . BREAK block is the last iteration of that block.

Syntax

```
LAST ( break-group )
```

break-group

The name of a field or expression you named in the block header with the BREAK BY option.

Example

The first FOR EACH block produces a list of the on hand values of the items in inventory. It also automatically generates a total of these on hand values.

The second FOR EACH block does exactly the same thing, except Progress does not generate the total. Instead, the procedure uses the ACCUMULATE statement and the LAST function. Thus, you can substitute your own labels and formats for the grand total.

r-last.p

```
FOR EACH item BY on-hand * price DESCENDING:
     DISPLAY item-num on-hand * price (TOTAL) LABEL "Value-oh"
         WITH USE-TEXT.
END.

FOR EACH item BREAK BY on-hand * price DESCENDING:
     FORM item.item-num value-oh AS DECIMAL
          LABEL "Value-oh" WITH COLUMN 40 USE-TEXT.
     DISPLAY item-num on-hand * price @ value-oh.
     ACCUMULATE on-hand * price (TOTAL).
     IF LAST(on-hand * price) THEN DO:
        UNDERLINE value-oh.
        DISPLAY ACCUM TOTAL on-hand * price @ value-oh.
     END.
END.
```

See also

FIRST function, FIRST-OF function, LAST-OF function

LASTKEY function

Returns the integer key code of the most recent event read from the user (that is, from the keyboard or mouse) during an interaction with a procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
LASTKEY
```

Example

In this procedure, the user can move through the customer file and update certain fields in each of the customer records. The GO-ON option tells the procedure to continue on to the following statements if the user presses **F9**, **F10**, or **F12**. To determine what action to take, the LASTKEY function compares the key code of the last key pressed with the key codes **F9**, **F10**, and **F12**.

r-lastky.p

```
DISPLAY "You may update each customer.  After making your changes,"
  SKIP "Press one of:" SKIP(1)
  KBLABEL("GO") "Make the changes permanent"  SKIP
  KBLABEL("END-ERROR")  "Undo changes and exit"       SKIP
  "F9" SPACE(7) "Undo changes and try again"  SKIP
  "F10" SPACE(6) "Find next customer"          SKIP
  "F12" SPACE(6) "Find previous customer"
  WITH CENTERED FRAME instr.

FIND FIRST customer.

REPEAT:
    UPDATE cust-num name address city state
          GO-ON(F9 F10 F12) WITH 1 DOWN.
    IF LASTKEY = KEYCODE("F9")
    THEN UNDO, RETRY.
    ELSE IF LASTKEY = KEYCODE("F10")
    THEN FIND NEXT customer.
    ELSE IF LASTKEY = KEYCODE("F12")
    THEN FIND PREV customer.
END.
```

Notes

- The LASTKEY function is double-byte enabled. The LASTKEY function returns values only after the input method places the data in the keyboard buffer. It returns the key code of the most recent key sequence returned from the keyboard buffer. A key sequence is the set of keystrokes necessary to generate one character or function key event in Progress.
- If you used a READKEY statement that timed out (you specified a number of seconds by using the PAUSE option with the READKEY statement), or if a PAUSE statement times out, the value of LASTKEY is -1.
- If you use the PAUSE option with the READKEY statement, the value of LASTKEY is the key you press to end the PAUSE.
- When Progress starts, the value of LASTKEY is -1. This value remains the same until the first input, READKEY, or procedure pause occurs. The LASTKEY function is reset to -1 each time you return to the Progress Editor.
- If you read data from a file, LASTKEY is set to the last character read from the file. For an INSERT, PROMPT-FOR, SET or UPDATE statement, this is always KEYCODE("RETURN"). For a READKEY statement, this is the character read from the file. If you reach past the end of the file, LASTKEY is -2.
- For more information on keys, see *OpenEdge Development: Programming Interfaces*.

See also READKEY statement

LAST-OF function

Returns a TRUE value if the current iteration of a DO, FOR EACH, or REPEAT . . . BREAK block is the last iteration for a particular value of a break group.

Syntax

```
LAST-OF ( break-group )
```

break-group

The name of a field or expression you named in the block header with the BREAK BY option.

Example

This procedure uses LAST-OF to display a single line of information on each cat-page group in the item file, without displaying any individual item data. It produces a report that shows the aggregate value on-hand for each catalog page.

r-lastof.p

```
FOR EACH item BREAK BY cat-page:
     ACCUMULATE on-hand * price (TOTAL BY cat-page).
     IF LAST-OF(cat-page)
     THEN DISPLAY cat-page (ACCUM TOTAL BY cat-page
                  on-hand * price) LABEL "Value-oh".
END.
```

See also

FIRST function, FIRST-OF function, LAST-OF function

LC function

Returns a character string identical to a specified string, but all uppercase letters in the string are converted to lowercase.

Syntax

```
LC ( string )
```

string

A character expression that contains uppercase letters you want to convert to lowercase.

Example

This procedure finds a customer record. After the user updates the sales-rep field, the procedure converts the first character of the sales-rep value to uppercase and the remaining characters to lowercase.

r-lc.p

```
REPEAT:
    PROMPT-FOR Customer.Cust-num.
    FIND Customer USING Cust-num.
    DISPLAY Name.
    UPDATE sales-rep.
    sales-rep = CAPS(SUBSTRING(sales-rep, 1, 1) ) +
                LC(SUBSTRING(Sales-rep, 2) ).
    DISPLAY Sales-rep.
END.
```

The CAPS function uses the SUBSTRING function to extract the first character of the field, which it then converts to uppercase.

In the LC function, the result of the SUBSTRING function is the remaining characters in the sales-rep field, starting with character position 2. (No length is specified, so the remainder of the string is assumed). The LC function converts these characters to lowercase.

Notes

- The LC function returns lowercase characters relative to the settings of the -cpinternal and -cpcase startup parameters. For more information on these parameters, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- The LC function is double-byte enabled. The specified expression can yield a string containing double-byte characters; however, the LC function changes only single-byte characters in the string.

See also CAPS function

LDBNAME function

Returns the logical name of a database that is currently connected.

Syntax

```
LDBNAME
  (
    {   integer-expression
      | logical-name
      | alias
      | BUFFER bufname
    }
  )
```

integer-expression

The sequence number of a database the OpenEdge session is connected to. For example, LDBNAME(1) returns information on the first database the OpenEdge session is connected to, LDBNAME(2) returns information on the second database the OpenEdge session is connected to, etc. If you specify a sequence number that does not correspond to a database the OpenEdge session is connected to, the LDBNAME function returns the Unknown value (?).

logical-name or *alias*

These forms of the LDBNAME function require a quoted character string or a character expression as a parameter. If the parameter is the logical name of a connected database or an alias of a connected database then the logical name is returned. Otherwise, Progress returns the Unknown value (?).

BUFFER *bufname*

The name of a database table or buffer. The BUFFER option lets you determine the database a certain table belongs to without hard-coding the logical database name or alias.

Example

This procedure disconnects all currently connected databases. After a database is disconnected, the connected databases are renumbered to reflect the change. For example, if databases 1, 2, 3, and 4, are connected and the procedure disconnects database 3, database 4 becomes database 3.

r-ldbnm.p

```
DO WHILE LDBNAME(1) <> ? : /* the parameter. is the number 1 */
  DISCONNECT VALUE(LDBNAME(1)).
END.
```

Note

To determine if a particular name is an ALIAS or a logical database name, use this following procedure:

r-tstnm.p

```
DEFINE VARIABLE testnm as char.
SET testnm.
IF LDBNAME (testnm) = testnm
  THEN MESSAGE testnm "is a true logical database name.".
  ELSE
    IF LDBNAME (testnm) = ?
      THEN MESSAGE
        testnm "is not the name or alias of any connected database.".
      ELSE MESSAGE
        testnm "is an ALIAS for database " LDBNAME(testnm).
```

See also

CONNECT statement, CONNECTED function, CREATE ALIAS statement, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, NUM-DBS function, PDBNAME function

LE or < = operator

Returns a TRUE value if the first of two expressions is less than or equal to the second.

Syntax

```
expression { LE | <= } expression
```

expression

A constant, field name, variable name, or expression. The expressions on either side of the LE or < = must be of the same data type, although one can be integer and the other decimal.

Example

This procedure lists all the items with zero or negative on-hand quantities:

r-le.p

```
FOR EACH item WHERE on-hand <= 0:
    DISPLAY item.item-num item-name on-hand.
END.
```

Notes

- By default, Progress uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).
- If either of the expressions is the Unknown value (`?`), then the result is the Unknown value (`?`); if both of the expressions are the Unknown value (`?`), then the result is TRUE.

- You can compare character strings with LE. Most character comparisons are case insensitive in Progress. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *expression* is a field or variable defined as case sensitive, the comparison is case sensitive and “Smith” does not equal “smith”.

- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ’ , but lowercase a is greater than these.

- You can use LE to compare DATE, DATETIME, and DATETIME-TZ data. The data type that contains less information (that is, a DATE value contains less information than a DATETIME value, and a DATETIME value contains less information than a DATETIME-TZ value) is converted to the data type with more information by setting the time value to midnight, and the time zone value to the session's time zone (when the data type does not contain the time or time zone). Comparisons with DATETIME-TZ data are based on Coordinated Universal Time (UTC) date and time.

- You can use LE to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or Progress raises a run-time error.

- You cannot use LE to compare one CLOB field to another.

LEAVE statement

Exits from a block. Execution continues with the first statement after the end of the block.

Syntax

```
LEAVE [ label ]
```

label

The name of the block you want to leave. If you do not name a block, Progress leaves the innermost iterating block that contains the LEAVE statement. If there is no such block, then Progress leaves the procedure block.

Example

This procedure represents part of a menu program. If the user chooses N, P, F, or Q, the procedure leaves the inner choose block and goes on to process the menu selection. If the user presses any other key, the procedure rings the terminal bell.

r-leave.p

```
DEFINE VARIABLE valid-choice AS CHARACTER INITIAL "NPFQ".
DEFINE VARIABLE selection AS CHARACTER FORMAT "x".

main-loop:
REPEAT:
  choose:
  REPEAT ON ENDKEY UNDO choose, RETURN:
    MESSAGE "(N)ext (P)rev (F)ind (Q)uit"
      UPDATE selection AUTO-RETURN.
    IF INDEX(valid-choice, selection) <> 0
    THEN LEAVE choose. /* Selection was valid */
    BELL.
  END.  /* choose */

/* Processing for menu choices N, P, F here */

  IF selection = "Q" THEN LEAVE main-loop.
END.
```

See also

NEXT statement, RETURN statement, UNDO statement

LEFT-TRIM function

Removes leading white space, or other specified characters, from a CHARACTER or LONGCHAR expression.

Syntax

```
LEFT-TRIM ( expression [ , trim-chars ] )
```

expression

An expression (a constant, field name, variable name, or expression) whose value is a CHARACTER or LONGCHAR. If *expression* is a case-sensitive variable, Progress performs a case-sensitive trim. If *expression* is a LONGCHAR, the result is in the same code page.

trim-chars

A character expression that specifies the characters to be trimmed from *expression*. If you do not specify *trim-chars*, the LEFT-TRIM function removes spaces, tabs, line feeds, and carriage returns.

Example The following example shows the effect of the TRIM, LEFT-TRIM, and RIGHT-TRIM functions on a string value:

r-ltrim.p

```
DEFINE BUTTON b_left LABEL "Left Trim".
DEFINE BUTTON b_right LABEL "Right Trim".
DEFINE BUTTON b_trim  LABEL "Trim".
DEFINE BUTTON b_quit  LABEL "Quit" AUTO-ENDKEY.
DEFINE VARIABLE i AS INTEGER NO-UNDO.
DEFINE VARIABLE txt AS CHARACTER FORMAT "X(26)" INIT
  "***** This is a test *****".

DEFINE FRAME butt-frame
  txt i LABEL "String Length" SKIP(2)
  b_left b_right b_trim b_quit
WITH CENTERED TITLE "Original Text String".

DEFINE FRAME trimed-frame
  txt LABEL "Trimed Text"
  i   LABEL "Length"
WITH CENTERED.

ON CHOOSE OF b_trim, b_right, b_left IN FRAME butt-frame
DO:
  FRAME trimed-frame:TITLE  = "Data After " + SELF:LABEL.
  DISPLAY TRIM(txt, "* ") WHEN SELF:LABEL = "Trim" @  txt
    LENGTH(TRIM(txt, "* ")) WHEN SELF:LABEL = "Trim" @ i
    LEFT-TRIM(txt,"* ") WHEN SELF:LABEL = "Left Trim"  @ txt
    LENGTH(LEFT-TRIM(txt,"* ")) WHEN SELF:LABEL = "Left Trim" @ i
    RIGHT-TRIM(txt, "* ") WHEN SELF:LABEL = "Right Trim" @ txt
    LENGTH(RIGHT-TRIM(txt, "* ")) WHEN SELF:LABEL = "Right Trim"  @ i
  WITH FRAME trimed-frame.
END.

ENABLE b_left b_right b_trim b_quit WITH FRAME butt-frame.

i = LENGTH(txt).
DISPLAY txt i WITH FRAME butt-frame.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.
```

Notes

- The LEFT-TRIM function is similar to the TRIM function except that it trims characters only from the left end of the string.
- If *expression* is a case-sensitive field or variable, then *trim-chars* is also as case sensitive. Otherwise, *trim-chars* is not case sensitive.
- The LEFT-TRIM function is double-byte enabled. The specified *expression* and *trim-chars* arguments can contain double-byte characters. LEFT-TRIM does not remove double-byte space characters by default.

See also RIGHT-TRIM function, TRIM function

LENGTH function

Returns the number of characters, bytes, or columns in a string. Returns the number of bytes in an expression of type RAW or a BLOB field.

Syntax

```
LENGTH ( { string [ , type ] | raw-expression | blob-field } )
```

string

A character expression. The specified *string* can be a character string, a CLOB field, or a LONGCHAR variable. and may contain double-byte characters.

type

A character expression that indicates whether you want the length of *string* in character units, bytes, or columns. A double-byte character registers as one character unit. By default unit of measurement is character units.

There are three valid types: "CHARACTER," "RAW," and "COLUMN." The expression "CHARACTER" indicates that the length is measured in characters, including double-byte characters. The expression "RAW" indicates that the length is measured in bytes. The expression "COLUMN" indicates that the length is measured in display or print character-columns. If you specify the *type* as a constant expression, Progress validates the type specification at compile time. If you specify the *type* as a non-constant expression, Progress validates the type specification at run time.

Note: The expression "COLUMN" is not valid for a LONGCHAR variable or a CLOB field.

raw-expression

A function or variable name that returns a raw value.

blob-field

An expression that evaluates to a BLOB field.

Examples

This procedure produces a report that contains item information. Because the information on the report fills the entire width of the screen, this procedure shortens the information in the description field for each item. If the description of an item is longer than eight characters, the procedure converts the description to the first eight characters followed by ellipses.

r-length.p

```
DEFINE VARIABLE short-name AS CHARACTER
    FORMAT "x(11)" LABEL "Desc".

FOR EACH item:
    IF LENGTH(item-name, "CHARACTER") > 8 THEN
        short-name = SUBSTRING(item-name,1,8, "FIXED") + "..." .
    ELSE short-name = item-name.
    DISPLAY item-num short-name on-hand allocated
            re-order on-order price FORMAT "$>>>9.99".
END.
```

In this procedure, the LENGTH function returns the number of bytes in the name of number 29. The procedure returns a 15, the number of bytes in the name, `Bug in a Rug-by`.

r-rawlen.p

```
DEFINE VARIABLE i AS INTEGER.

FIND customer WHERE Cust-num = 29.
i = LENGTH(name, "RAW").
DISPLAY Name i LABEL "Byte Length".
```

Note

If the value of the expression is the Unknown value (?), the LENGTH function returns the Unknown value (?).

LENGTH statement

Changes the number of bytes in a raw variable.

Syntax

```
LENGTH ( variable ) = expression
```

variable

A variable of type RAW.

expression

An expression that returns an integer.

Example

This procedure takes the number of bytes in the name stored in the variable r1 and truncates it to 2 bytes:

r-rawln1.p

```
/* You must connect to a non-PROGRESS demo database to run
  this procedure */

DEFINE VARIABLE r1 as RAW.

FIND customer WHERE cust-num = 29.
r1 = RAW(name).
LENGTH(r1) = 2.
```

Notes

- If *variable* is the Unknown value (?), it remains the Unknown value (?).
- If *expression* is greater than the number of bytes in *variable*, Progress appends null bytes so that the length of *variable* equals the length of *expression*.

LIBRARY function

Parses a character string in the form *path-name<<member-name>>*, where *path-name* is the pathname of a Progress r-code library and *member-name* is the name of a file within the library, and returns the pathname of the library. The double angle brackets indicate that *member-name* is a file in a library. If the string is not in this form, the LIBRARY function returns the Unknown value (?).

Typically, you use the LIBRARY function with the SEARCH function to retrieve the name of a library. The SEARCH function returns character strings of the form *path-name<<member-name>>* if it finds a file in a library.

Syntax

```
LIBRARY ( string )
```

string

A character expression whose value is the pathname of a file in a library.

Example

This procedure searches for a file that you specify. It displays a message indicating whether the file is not found in your path, is found in a library within your path, or is found in your path but not in a library.

r-rlib.p

```
DEFINE VARIABLE what-lib AS CHARACTER.
DEFINE VARIABLE location AS CHARACTER.
DEFINE VARIABLE myfile   AS CHARACTER FORMAT "x(16)" LABEL "R-code File".

SET myfile.
location = SEARCH(myfile).

IF location = ?
THEN DO:
   MESSAGE "Can't find" myfile.
   LEAVE.
END.

what-lib = LIBRARY(location).
IF what-lib <> ?
THEN MESSAGE myfile "can be found in library" what-lib.
ELSE MESSAGE myfile "is not in a library but is in" location.
```

Note

You can improve the performance of an application by using the SEARCH and LIBRARY functions to build absolute or relative pathnames for the files you want to execute several times with the RUN statement. Passing full or relative pathnames to the RUN statement avoids the need to search the PROPATH each time.

See also

MEMBER function, SEARCH function

LINE-COUNTER function

Returns the current line number of paged output.

The initial value of LINE-COUNTER is 1. At the completion of each DISPLAY statement, Progress increments LINE-COUNTER by the number of lines that were output in that DISPLAY statement. LINE-COUNTER continues to increase until after at least one line has been printed on a new page.

LINE-COUNTER returns a 0 if the output is not paged.

Syntax

```
LINE-COUNTER [ ( stream ) ]
```

stream

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. For more information on streams, see this book's DEFINE STREAM statement reference entry and *OpenEdge Development: Programming Interfaces*.

Example

This procedure prints a customer report, categorized by state. At the end of each state category, it tests to see if there are at least four lines left on the page. The LINE-COUNTER function returns the current line number of output. If that number plus four is greater than the total number of lines on the page (returned by the PAGE-SIZE function), then the procedure starts the new page. If there are four or more lines left, the procedure skips a line before printing the next customer record.

r-linec.p

```
OUTPUT TO PRINTER.
FOR EACH customer BREAK BY state:
    DISPLAY cust-num name address city state.
    IF LAST-OF(state) THEN DO:
        IF LINE-COUNTER + 4 > PAGE-SIZE
        THEN PAGE.
        ELSE DOWN 1.
    END.
END.
```

Notes

- When output is sent to a device other than the terminal screen, Progress defers displaying a frame until another frame is displayed. That way, if you display the same frame several times consecutively, Progress performs all those displays at once. Because of this optimization, if the last display fills the page, the value returned by the LINE-COUNTER function can be larger than the page size, even though the next frame is displayed at the start of the new page.

- Use a procedure like this one to verify that output is positioned on the first non-header line of a new page:

```
DEFINE VARIABLE newpage AS LOGICAL INITIAL YES.
DEFINE STREAM output1.

FOR EACH customer:
  FORM HEADER "Page Header" PAGE-NUMBER(output1)
    "Line" LINE-COUNTER(output1)
    WITH FRAME one PAGE-TOP NO-LABELS NO-BOX.

  VIEW STREAM output1 FRAME one.
  DISPLAY STREAM output1 name PAGE-NUMBER(output1)
    LINE-NUMBER(output1) WITH NO-LABELS NO-BOX.

  IF new-page THEN DISPLAY STREAM output1 "First Line".
  IF LINE-COUNTER(output1) > PAGE-SIZE(output1)
  THEN newpage = YES.
  ELSE newpage = NO.
END.
```

See also DEFINE STREAM statement

LIST-EVENTS function

Returns a comma-separated list of the valid events for a specified object or widget.

Note: Does not apply to SpeedScript programming.

Syntax

```
LIST-EVENTS ( widget-handle [ , platform ] )
```

widget-handle

A handle to a valid object or widget. The function returns a list of the events that are valid for that object or widget.

platform

A character-string value that specifies a display type. Valid values are GUI and TTY. Some events are valid only on certain platforms. If you omit the *platform* parameter, Progress uses the platform for the current session.

Example

The following example uses the LIST-EVENTS function to populate a selection list with all the valid events for a widget. When you run this procedure, type **?** at any time to see a list of valid events for the widget that currently has focus.

r-levent.p

```
DEFINE VARIABLE inv-price LIKE item.price.
DEFINE VARIABLE inv-value LIKE item.price.
DEFINE VARIABLE report-type AS INTEGER INITIAL 1.

DEFINE VARIABLE event-list AS CHARACTER VIEW-AS SELECTION-LIST
                           INNER-CHARS 20 INNER-LINES 5
                           SCROLLBAR-VERTICAL.

DEFINE BUTTON ok-butt LABEL "OK" AUTO-GO.
DEFINE BUTTON cancel-butt LABEL "CANCEL" AUTO-ENDKEY.

FORM
   inv-price LABEL "Price"
      AT ROW 1.25 COLUMN 2
   report-type LABEL "Report Sorted ..."
      AT ROW 2.25 COLUMN 2
      VIEW-AS RADIO-SET RADIO-BUTTONS "By Catalog Page",   1,
                                      "By Inventory Value", 2
   SKIP
   ok-butt cancel-butt
   WITH FRAME select-frame SIDE-LABELS.

FORM
   event-list
   WITH FRAME list-frame NO-LABELS TITLE "Events" WIDTH 30.

ON ? ANYWHERE
  DO:
     FRAME list-frame:TITLE = "Events for " + FOCUS:TYPE.
     event-list:LIST-ITEMS IN FRAME list-frame = LIST-EVENTS(FOCUS).
     DISPLAY event-list WITH FRAME list-frame.
     ENABLE event-list WITH FRAME list-frame.
     RETURN NO-APPLY.
END.

ENABLE ALL WITH FRAME select-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

See also

LAST-EVENT system handle, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function, LIST-WIDGETS function, VALID-EVENT function

LIST-QUERY-ATTRS function

Returns a comma-separated list of attributes and methods that are supported for an object or widget.

Syntax

```
LIST-QUERY-ATTRS ( widget-handle )
```

widget-handle

A handle to a valid object or widget. The function returns a list of the attributes and methods that are supported for that object or widget.

Example

The following example uses the LIST-QUERY-ATTRS and LIST-SET-ATTRS functions to populate selection lists with the valid attributes and methods for a specified widget. When you run this procedure, type **?** at any time to see lists of valid attributes for the widget that currently has focus.

r-lattrs.p

```
DEFINE VARIABLE inv-price LIKE item.price.
DEFINE VARIABLE inv-value LIKE item.price.
DEFINE VARIABLE report-type AS INTEGER INITIAL 1.

DEFINE VARIABLE qattr-list AS CHARACTER LABEL "Readable"
                   VIEW-AS SELECTION-LIST INNER-CHARS 20 INNER-LINES 5
                   SCROLLBAR-VERTICAL SORT.
DEFINE VARIABLE wattr-list AS CHARACTER LABEL "Writable"
                   VIEW-AS SELECTION-LIST INNER-CHARS 20 INNER-LINES 5
                   SCROLLBAR-VERTICAL SORT.
DEFINE BUTTON ok-butt LABEL "OK" AUTO-GO.
DEFINE BUTTON cancel-butt LABEL "CANCEL" AUTO-ENDKEY.

FORM
   inv-price LABEL "Price"
      AT ROW 1.25 COLUMN 2
   report-type LABEL "Report Sorted ..."
      AT ROW 2.25 COLUMN 2
      VIEW-AS RADIO-SET RADIO-BUTTONS "By Catalog Page",    1,
                                      "By Inventory Value", 2
   SKIP
   ok-butt cancel-butt
   WITH FRAME select-frame SIDE-LABELS.

FORM
   qattr-list wattr-list
   WITH FRAME list-frame TITLE "Attributes" WIDTH 30 COLUMN 47.

ON ? ANYWHERE
  DO:
     FRAME list-frame:TITLE = "Attributes for " + FOCUS:TYPE.
     qattr-list:LIST-ITEMS IN FRAME list-frame = LIST-QUERY-ATTRS(FOCUS).
     wattr-list:LIST-ITEMS IN FRAME list-frame = LIST-SET-ATTRS(FOCUS).
     ENABLE qattr-list wattr-list WITH FRAME list-frame.
     RETURN NO-APPLY.
END.

ENABLE ALL WITH FRAME select-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

See also

CAN-QUERY function, CAN-SET function, LAST-EVENT system handle, LIST-EVENTS function, LIST-SET-ATTRS function, LIST-WIDGETS function, VALID-EVENT function

LIST-SET-ATTRS function

Returns a comma-separated list of attributes that can be set for an object or widget.

Syntax

```
LIST-SET-ATTRS ( widget-handle )
```

widget-handle

A handle to a valid object or widget. The function returns a list of the attributes that can be set for that object or widget.

Example

For an example of the LIST-SET-ATTRS function, see the LIST-QUERY-ATTRS function reference entry.

See also

CAN-QUERY function, CAN-SET function, LAST-EVENT system handle, LIST-EVENTS function, LIST-QUERY-ATTRS function, LIST-WIDGETS function, VALID-EVENT function

LIST-WIDGETS function

Returns a comma-separated list of objects and widget types that respond to a specified event.

Note: Does not apply to SpeedScript programming.

Syntax

```
LIST-WIDGETS ( event-name [ , platform ] )
```

event-name

A character-string expression that evaluates to an event name.

platform

A character-string value that specifies a display type. Valid values are GUI and TTY. Some events are valid only on certain platforms. If you omit the *platform* parameter, Progress uses the platform for the current session.

Example

The following example prompts for an event name and then displays a list of widget types that support that event:

r-lwids.p

```
DEFINE VARIABLE event-name AS CHARACTER FORMAT "x(24)" LABEL "Event".
DEFINE VARIABLE widget-list AS CHARACTER LABEL "Widgets"
                VIEW-AS SELECTION-LIST INNER-CHARS 24 INNER-LINES 6
                 SCROLLBAR-VERTICAL.

FORM
   event-name SKIP
   widget-list
   WITH FRAME main-frame SIDE-LABELS.

REPEAT WITH FRAME main-frame:
   DISABLE widget-list.
   SET event-name.
   widget-list:LIST-ITEMS = LIST-WIDGETS(event-name).
   DISPLAY widget-list.
   ENABLE widget-list.
   PAUSE.
END.
```

See also

LAST-EVENT system handle, LIST-EVENTS function, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function, VALID-EVENT function

LOAD statement (Windows only)

Creates application defaults, involving colors, fonts, environment variables, etc., or loads existing defaults, to a graphical or character application. Specifically, LOAD:

- Creates registry keys and initialization file entries.
- Creates new initialization files.
- Loads entries from the registry or from an existing initialization file.

For more information on application defaults, see the chapter on colors and fonts in *OpenEdge Development: Programming Interfaces*.

Note: Does not apply to SpeedScript programming.

Syntax

```
LOAD environment
  [ DIR directory ]
  [ APPLICATION ]
  [ NEW ]
  [ BASE-KEY { key-name | "INI" } ]
  [ NO-ERROR ]
```

environment

A CHARACTER expression that evaluates to one of the following:

- The name of a registry key to create.
- The name of an initialization file to create.
- The name of an existing registry key.
- The name of an existing initialization file.

DIR *directory*

A CHARACTER expression that evaluates to the directory path of one of the following:

- An existing initialization file.
- An initialization file to create.

If you omit this option, LOAD looks for an existing initialization file, or creates a new initialization file, in the working directory.

APPLICATION

Has no effect; supported only for backward compatibility.

NEW

Creates a new registry key or a new initialization file. If the key or file already exists, LOAD overwrites its data.

BASE-KEY *key-name* | "INI"

Bypasses the standard search rules.

If you specify BASE-KEY *key-name*, LOAD looks for or creates the environment in the registry only under base key *key-name*.

If you specify BASE-KEY "INI" (the quotes are required), LOAD looks for or creates only the initialization file *environment*.

NO-ERROR

Suppresses error messages while LOAD executes. When LOAD finishes, you can learn what errors, if any, occurred by using the attributes and methods of the ERROR-STATUS handle. For more information on the ERROR-STATUS handle, see the "Handle Reference" section on page 1379.

Example

See the USE statement reference entry for an example.

Notes

- If you specify LOAD *environment*, LOAD searches for a registry key and for an existing initialization file, and tries to load one or the other. The search logic, which assumes that *environment* has the format *path**rootname*.*extension* (where *path* and *extension* are optional) and that *version* is the current Progress version, is as follows:

  a) Search the registry under HKEY_CURRENT_USER for *path**rootname*.*extension*.
  If found, load it.

  b) Else search the registry under HKEY_CURRENT_USER for *SOFTWARE**PSC**PROGRESS**version**path**rootname*.*extension*.
  If found, load it.

  c) Else search the registry under HKEY_CURRENT_USER for *SOFTWARE**path**rootname*.*extension*.
  If found, load it.

  d) Else search the registry under HKEY_CURRENT_USER for *rootname*.
  If found, load it.

  e) Else search the registry under HKEY_CURRENT_USER for *SOFTWARE**PSC**PROGRESS**version**rootname*.
  If found, load it.

  f) Else search the registry under HKEY_CURRENT_USER for *SOFTWARE**rootname*.
  If found, load it.

  g) Else search the registry under HKEY_LOCAL_MACHINE for *path**rootname*.*extension*.
  If found, load it.

  h) Else search the registry under HKEY_LOCAL_MACHINE for *SOFTWARE**PSC**PROGRESS**version**path**rootname*.*extension*.
  If found, load it.

  i) Else search the registry under HKEY_LOCAL_MACHINE for *SOFTWARE**path**rootname*.*extension*.
  If found, load it.

  j) Else search the registry under HKEY_LOCAL_MACHINE for *rootname*.
  If found, load it.

k) Else search the registry under HKEY_LOCAL_MACHINE for *SOFTWARE\PSC\PROGRESS\version\rootname.*
If found, load it.

l) Else search the registry under HKEY_LOCAL_MACHINE for *SOFTWARE\rootname.*
If found, load it.

m) Else search for the initialization file *path\rootname.extension*.
If found, load it.

n) Else, error.

- If you specify LOAD *environment* BASE-KEY *key-name*, where *key-name* is the name of a registry base key, LOAD loads the registry key *key-name\environment.*

  Registry base keys are as follows:

  – HKEY_CLASSES_ROOT

  – HKEY_CURRENT_CONFIG (Win95 and NT 4.0)

  – HKEY_CURRENT_USER

  – HKEY_DYN_DATA (Win95 and NT 4.0)

  – HKEY_LOCAL_MACHINE

  – HKEY_USERS

- If you specify LOAD *environment* BASE-KEY "INI," LOAD loads the initialization file *environment*.

- If you specify LOAD *environment* NEW, LOAD creates a new key in the registry under HKEY_CURRENT_USER and names the new key *environment*.

- If you specify LOAD *environment* NEW BASE-KEY *key-name*, LOAD creates a new key in the registry under *key-name* and names the new key *environment*.
- If you specify LOAD *environment* NEW BASE-KEY "INI," LOAD creates a new initialization file and names it *environment.ini*.
- To change the application environment, load defaults using the LOAD statement, make them current using the USE statement, then access them using the GET-KEY-VALUE and PUT-KEY-VALUE statements.

See also GET-KEY-VALUE statement, LOAD statement, PUT-KEY-VALUE statement, UNLOAD statement, USE statement

LOAD-PICTURE statement (Windows only; Graphical interfaces only)

Returns a COM-HANDLE to an OlePictureObject. You can use this COM-HANDLE to set graphical properties of controls.

Note: Does not apply to SpeedScript programming.

Syntax

```
LOAD-PICTURE [ image ]
```

image

A CHARACTER expression representing the name of the graphical file. This file can have one of the following extensions: .BMP, .WMF, .EMF, .ICO, .CUR, .DIB. If the filename is not fully qualified, LOAD-PICTURE searches for a matching file on the user's path.

Example

The following program fragment illustrates the use of the LOAD-PICTURE statement:

```
DEFINE VAR chPic as COM-HANDLE.
DEFINE VAR chControl as COM-HANDLE.

/* Get the COM-HANDLE of an Image control */

/* Use LOAD-PICTURE to get a COM-HANDLE to the bitmap contained in myPic.bmp */
chPic = LOAD-PICTURE("myPic.bmp").

/* Set the control's Picture property for the bitmap */
chControl:Picture = chPic.
```

LOCKED function

Returns a TRUE value if a record is not available to a prior FIND . . . NO-WAIT statement because another user has locked a record.

Syntax

```
LOCKED record
```

record

The name of a record or buffer.

To use the LOCKED function with a record in a file defined for multiple databases, you must qualify the record's filename with the database name. See the Record phrase reference entry for more information.

Example

The FIND statement in this procedure tries to retrieve a customer record according to a supplied customer number. Because of the NO-ERROR option, the FIND statement does not return an error if it cannot find the record. The NO-WAIT option causes FIND to return immediately if the record is in use by another user.

r-locked.p

```
REPEAT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING customer.cust-num NO-ERROR NO-WAIT.
    IF NOT AVAILABLE customer THEN DO:
        IF LOCKED customer
        THEN MESSAGE "Customer record is locked".
        ELSE MESSAGE "Customer record was not found".
        NEXT.
    END.
    DISPLAY cust-num name city state.
END.
```

A record might not be available if it is locked (being used by another user) or does not exist. The LOCKED function returns a TRUE value if the record is locked. In this case, the `r-locked.p` procedure displays a message that the record is locked. If the record is not locked, the procedure displays a message that the record does not exist.

Note: The result of the LOCKED function depends on the lock mode specified. For example, if the FIND statement uses SHARE-LOCK and no user has an EXCLUSIVE-LOCK on the record, the LOCKED function returns FALSE. If the FIND statement uses SHARE-LOCK and another user has an EXCLUSIVE-LOCK on the record, the LOCKED function returns TRUE. The current copy of the record in the buffer to which the LOCKED function applies is not locked. Rather, LOCKED refers to an error condition that can occur if the record is locked. Consider using the NO-WAIT and NO-ERROR options on the FIND statement to return immediately and raise an error condition.

See also AMBIGUOUS function, AVAILABLE function, FIND statement, NEW function

LOG function

Calculates the logarithm of an expression using a specified base.

Syntax

```
LOG ( expression [ , base ] )
```

expression

A decimal expression that you want the logarithm of.

base

A numeric expression that is the base you want to use. If you do not specify a base, LOG returns the natural logarithm, base (e). The *base* must be greater than 1.

Example

This procedure prompts the user for a base and a number, and then displays the log of the number. The VALIDATE option on the UPDATE statement ensures that the user enters a base value greater than 1 and a number greater than 0.

r-log.p

```
DEFINE VARIABLE base AS DECIMAL FORMAT ">>>,>>>.9999".
DEFINE VARIABLE number AS DECIMAL.

REPEAT:
  UPDATE base
    VALIDATE(base > 1, "Base must be greater than 1").
  REPEAT:
    UPDATE number
      VALIDATE(number > 0, "Number must be positive").
    DISPLAY number LOG(number, base)
      LABEL "LOG(NUMBER, BASE)".
  END.
END.
```

Notes

- The LOG function is accurate to approximately 10 decimal places.
- After converting the base and exponent to floating-point format, the LOG function uses standard system routines. On some machines, the logarithm routines do not handle large numbers well and might cause your terminal to hang.

LOGICAL function

Converts any data type into the LOGICAL data type.

Syntax

```
LOGICAL (<expression> [, <char-expression-format>]  )
```

expression

An *expression* in the data type that you want to convert to logical.

char-expression-format

A character expression that evaluates to a valid logical format, such as "si/no", or "da/nyet". This argument is ignored unless <*expression*> is of CHARACTER type. Also, this argument is only needed if <*expression*> evaluates to something other than the usual TRUE or FALSE, or YES or NO values.

Example

The following code fragment illustrates the Logical function:

```
DEFINE VARIABLE mychar AS CHAR.
DEFINE VARIABLE v-log AS LOGICAL.
mychar="si".
v-log=LOGICAL (mychar, "si/no")
/*v-log is TRUE*/
```

If the value of <*expression*> is the Unknown value (?), the LOGICAL function returns the Unknown value (?).

If <*expression*> is of type DECIMAL, INTEGER, DATE, or HANDLE, the function returns TRUE if the value of <*expression*> is nonzero. If the value of <*expression*> is 0, it returns FALSE. The second argument is ignored if present.

If <*expression*> is of type CHARACTER, it returns TRUE or FALSE depending on the value in the expression and the format used. Whether or not <*char-expression-format*> is given, the case-insensitive values TRUE, FALSE, YES, NO, abbreviated to 1 character, are always accepted. For example, a "Y" is interpreted as TRUE.

If <*char-expression-format*> is given, it is validated. If it is not valid, an error message appears and the Unknown value (?) is returned. Otherwise, the format is used to interpret the character string if it is not one of the following: TRUE, FALSE, YES, or NO. For example, LOGICAL ("si", "si/no") returns TRUE.

Data types such as RAW, MEMPTR, LVARBINARY, and so on return the Unknown value (?), but this is not considered an error.

See also STRING function, INTEGER function, DATE function

Logical values

Represent values of logical expressions.

Syntax

```
{ [ YES | TRUE ] | [ NO | FALSE ] }
```

YES | TRUE

A value that signifies a valid result for a logical expression.

NO | FALSE

A value that signifies an invalid result for a logical expression.

Note

You must use these values in a procedure even if alternate values are given in the FORMAT specification for a field or variable.

LOOKUP function

Returns an integer giving the position of an expression in a list. Returns a 0 if the expression is not in the list.

Syntax

```
LOOKUP ( expression , list [ , character ] )
```

expression

A constant, field name, variable name, or expression that results in a character value that you want to look up within a list of character expressions. If the value of *expression* is the Unknown value (?), the result of the LOOKUP function is the Unknown value (?).

list

A character expression that contains the expression you name with the *expression* argument. Each entry in the list is separated with a delimiter. The list can be a variable of type CHARACTER or LONGCHAR. If *list* contains the Unknown value (?), LOOKUP returns the Unknown value (?).

character

A delimiter you define for the list. The default is a comma. This allows the LOOKUP function to operate on non-comma-separated lists.

Examples

This procedure prompts the user for a New England state. The LOOKUP function tests the value against the list of states stored in the stlist variable. If there is no match (the result is 0), the procedure displays a message. Otherwise, the procedure prompts the user for another New England state.

r-lookup.p

```
DEFINE VARIABLE stlist AS CHARACTER
  INITIAL "ME,MA,VT,RI,CT,NH".
DEFINE VARIABLE state AS CHARACTER FORMAT "x(2)".

REPEAT:
  SET state LABEL
    "Enter a New England state, 2 characters".
  IF LOOKUP(state, stlist) = 0
  THEN MESSAGE "This is not a New England state".
END.
```

The following example uses a different delimiter, which list all fields that have "sls" or "sales" as words in their standard Dictionary labels:

r-look2.p

```
FOR EACH _Field
  WHERE LOOKUP("sls",_Label," ") > 0
  OR LOOKUP("sales",_Label," ") > 0:

  DISPLAY _Field-name _Label.
END.
```

Notes

- If *expression* contains a delimiter, LOOKUP returns the beginning of a series of entries in *list*. For example, LOOKUP("a,b,c","x,a,b,c") returns a 2.
- Most character comparisons are case insensitive in Progress. By default, upper-case and lower-case characters have the same sort value. However, you can define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If the *expression* or *list* is defined as case sensitive, the comparison between them is also case sensitive and "Smith" does not equal "smith".
- The LOOKUP function is double-byte enabled. The specified *expression* can yield a string value that contains double-byte characters and the *character* delimiter can be a double-byte character.

See also

ENTRY function, ENTRY statement, INDEX function

LT or < operator

Returns a TRUE value if the first of two expressions is less than the second.

Syntax

```
expression { LT | < } expression
```

expression

A constant, field name, variable name, or expression. The expressions on either side of the LT or < = must be of the same data type, although one can be an integer and the other can be a decimal.

Example

This procedure displays information for those item records whose on-hand value is less than the allocated value:

r-lt.p

```
FOR EACH item WHERE on-hand < allocated:
  DISPLAY item.item-num item-name on-hand allocated.
END.
```

Notes

- By default, Progress uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (-cpcoll) startup parameter take precedence over a collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. If you do not specify a collation with the -cpcoll startup parameter, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the default name "basic" (which might or might not exist in the convmap.cp file).
- If either of the expressions is the Unknown value (?), then the result is the Unknown value (?); if both of the expressions are the Unknown value (?), then the result is FALSE.
- You can compare character strings with LT. Most character comparisons are case insensitive in Progress. That is, upper-case and lower-case characters have the same sort value. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *expression* is a field or variable defined as case sensitive, the comparison is case sensitive and “Smith” does not equal “smith.”

- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ` , but lowercase a is greater than these.
- You can use LT to compare DATE, DATETIME, and DATETIME-TZ data. The data type that contains less information (that is, a DATE value contains less information than a DATETIME value, and a DATETIME value contains less information than a DATETIME-TZ value) is converted to the data type with more information by setting the time value to midnight, and the time zone value to the session's time zone (when the data type does not contain the time or time zone). Comparisons with DATETIME-TZ data are based on Coordinated Universal Time (UTC) date and time.
- You can use LT to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or Progress raises a run-time error.
- You cannot use LT to compare one CLOB field to another.

MATCHES operator

Compares a character expression to a pattern and evaluates to a TRUE value if the expression satisfies the pattern criteria.

Syntax

```
expression MATCHES pattern
```

expression

A CHARACTER or LONGCHAR expression that you want to check to see if it conforms with the *pattern*.

pattern

A character expression that you want to match with the string. This can include a constant, field name, variable name, or expression whose value is a character.

The *pattern* can contain wildcard characters: a period (.) in a particular position indicates that any single character is acceptable in that position; an asterisk (*) indicates that any group of characters is acceptable, including a null group of characters.

Example

This procedure displays customer information for all customers whose address ends in St. The procedure does not use an index for the customer search in `r-match.p`.

r-match.p

```
FOR EACH customer WHERE address MATCHES("*St"):
  DISPLAY name address city state country.
END.
```

Notes

- MATCHES does not use index information when performing a comparison; it always scans the entire data table.
- MATCHES does not ignore trailing blanks as does the equal (EQ) comparison operator. Thus, “abc” does not match “abc “ although they are considered equal.
- Most character comparisons are case insensitive in Progress. By default, all characters are converted to uppercase prior to comparisons. However, you can define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If the *expression* preceding the MATCHES keyword is a field or variable defined as case sensitive, the comparison is case sensitive. In a case-sensitive comparison “SMITH” does not equal “Smith”.
- MATCHES converts a LONGCHAR variable value to `-cpinternal` prior to comparison. The variable must convert without error, or Progress raises a run-time error.
- You cannot use MATCHES to compare one CLOB field to another.
- If you want to specify a period (.) or an asterisk (a constant, field name, variable name, or expression whose value is character)(*) as a literal character rather than a wildcard character in the pattern, enter a tilde (~) before the character. For example, the result of “*a.b” MATCHES “~*a~.b”is TRUE. If you specify the match pattern as a literal quoted string in a procedure file, enter each tilde as a double tilde (~ ~) so that they are interpreted as tildes for the match pattern.
- The MATCHES function is double-byte enabled. Both the specified *expression* and *pattern* arguments can contain double-byte characters.

See also BEGINS operator

MAXIMUM function

Compares two or more values and returns the largest value.

Syntax

```
MAXIMUM ( expression , expression [ , expression ] ... )
```

expression

A constant, field name, variable name, or expression. If there is a mixture of decimal and integer data types, decimal type is returned.

Example

In this procedure, if the credit-limit value is under 20,000, the procedure adds 10,000 to that value. Otherwise, the procedure sets credit-limit to 30,000. The MAXIMUM function determines the greater of the original credit-limit value and the new cred-lim2 value.

r-maxmum.p

```
DEFINE VARIABLE cred-lim2 AS DECIMAL
    FORMAT ">>,>>9.99".FOR EACH customer:
    IF credit-limit < 20000 THEN cred-lim2 = credit-limit + 10000.
    ELSE cred-lim2 = 30000.
    DISPLAY credit-limit cred-lim2
      MAXIMUM(cred-lim2, credit-limit)
      LABEL "Maximum of these two values".
END.
```

Notes

- When comparing character values, if at least one of the character fields is defined as case sensitive, then MAXIMUM treats all of the values as case sensitive for the sake of the comparisons. If none of the values is case sensitive, MAXIMUM treats lowercase letters as if they were uppercase letters.
- You can use MAXIMUM to compare DATE, DATETIME, and DATETIME-TZ data. The data type that contains less information (that is, a DATE value contains less information than a DATETIME value, and a DATETIME value contains less information than a DATETIME-TZ value) is converted to the data type with more information by setting the time value to midnight, and the time zone value to the session's time zone (when the data type does not contain the time or time zone). Comparisons with DATETIME-TZ data are based on Coordinated Universal Time (UTC) date and time.

See also

MINIMUM function

MD5-DIGEST function

Hashes the specified data using the RSA Message Digest Hash Algorithm (MD5), and returns a 16-byte binary message digest value as a RAW value.

Syntax

```
MD5-DIGEST( data-to-hash [ , hash-key ] )
```

data-to-hash

The source data to hash. The data may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the data is a CHARACTER or LONGCHAR value, Progress converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value.

hash-key

An optional key value to use in the hash operation. The key may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the key is a CHARACTER or LONGCHAR value, Progress converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value. This key value is combined with the source data before the hash operation begins.

If the *hash-key* value contains a null character, the null character is included in the hash operation.

See also

SHA1-DIGEST function

MEMBER function

Parses a reference to a member of a Progress r-code library and returns the simple member name.

Syntax

```
MEMBER ( string )
```

string

A character expression (a constant, field name, variable or expression that results in a character value) whose value is the pathname of a file in an r-code library.

The MEMBER function parses a character string in the form *path-name<<member-name>>*, where *path-name* is the pathname of a library and *member-name* is the name of a file within the library, and returns *member-name*. The double angle brackets indicate that *member-name* is a file in a library. If the string is not in this form, the MEMBER function returns the Unknown value (?).

Use the MEMBER function with the SEARCH function to determine whether a file is in a library. If a data file is in a library, you must first extract the file from the library in order to read it. (See *OpenEdge Deployment: Managing 4GL Applications* for more information on extracting a file from a library.) The SEARCH function returns a character string in the form *path-name<<member-name>>* if it finds a file in a library.

Example

This procedure prompts for the name of a file. Using this value, the procedure searches for the file. If it does not find the file, it displays a message and ceases operation. If it does find the file, it tests to see if the file is in a library. If so, the procedure displays the filename and the name of the library. Otherwise, the procedure displays the pathname of the file returned by SEARCH.

r-memb.p

```
DEFINE VARIABLE what-lib AS CHARACTER.
DEFINE VARIABLE location AS CHARACTER.
DEFINE VARIABLE myfile   AS CHARACTER FORMAT "x(16)" LABEL "R-code File".

SET myfile.
location = SEARCH(myfile).

IF location = ?
THEN DO:
   MESSAGE "Can't find" myfile.
   LEAVE.
END.

what-lib = LIBRARY(location).

IF what-lib <> ?
THEN MESSAGE MEMBER(location) "can be found in library" what-lib.
ELSE MESSAGE myfile "is not in a library but is in" location.
```

See also

LIBRARY function, SEARCH function

MESSAGE statement

Displays messages in the message area at the bottom of the window or in an alert box (or in an output stream—see the Notes section). By default, an area at the bottom line of the window is reserved for Progress system messages. An area above that is reserved for messages you display with the MESSAGE statement.

Syntax

```
MESSAGE
  [ COLOR color-phrase ]
  { expression | SKIP [ ( n ) ] } ...
  [ VIEW-AS ALERT-BOX
      [ alert-type ]
      [ BUTTONS button-set ]
      [ TITLE title-string ]
  ]
  [ { SET | UPDATE } field
        { AS datatype | LIKE field }
        [ FORMAT string ]
        [ AUTO-RETURN ]
  ]
  [ IN WINDOW window ]
```

`COLOR color-phrase`

Displays a message using the color you specify with the COLOR phrase.

```
   NORMAL
|  INPUT
|  MESSAGES
|  protermcap-attribute
|  dos-hex-attribute
|  { [ BLINK- ] [ BRIGHT- ] [ fgnd-color ] [ bgnd-color ] }
|  { [ BLINK- ] [ RVV- ] [ UNDERLINE- ] [ BRIGHT- ]
        [ fgnd-color ] }
|  VALUE ( expression )
```

For more information on *color-phrase*, see the COLOR phrase reference entry.

Note: The COLOR phrase does not have any effect in a Windows environment.

expression

An expression (a constant, field name, variable name, or expression) whose value you want to display in the message area. If *expression* is not character, it is converted to character before it is displayed. If you do not use this option, you must use either the SET or UPDATE option.

`SKIP [ ( n ) ]`

Indicates a number (*n*) of blank lines to insert into the message. The value of *n* can be 0. If you do not specify *n*, or if *n* is 0, a new line is started unless the current position is already the start of a new line.

You can only use this option with the VIEW-AS ALERT-BOX option.

`VIEW-AS ALERT-BOX [ alert-type ]`

Specifies that the message is displayed in an alert box rather than in the window message area. The value of *alert-type* determines the type of alert box. The possible values are:

- MESSAGE
- QUESTION
- INFORMATION
- ERROR
- WARNING

The type of alert box affects the visual representation of the box.

`BUTTONS button-set`

Specifies what sets of buttons are available within the alert box. The possible button sets are as follows:

- YES-NO
- YES-NO-CANCEL
- OK
- OK-CANCEL
- RETRY-CANCEL

The name of each button set indicates the buttons in that set. For example, YES-NO contains two buttons labeled YES and NO; YES-NO-CANCEL contains three buttons labeled YES, NO, and CANCEL; OK contains a single button labeled OK. If you do not specify a button set, the default is OK.

`TITLE title-string`

Specifies a value to display in the title bar of the alert box.

`SET field`

Displays the *expression* you specified and SETs the field or variable you name. (It prompts the user for input and assigns the value entered to the field or variable.) You cannot test the field with the ENTERED function or the NOT ENTERED function.

UPDATE *field*

Displays the *expression* you specified and updates the field or variable you name. (It displays the current value of the field or variable, prompts for input, and assigns the value entered in the field or variable.) You cannot test the field with the ENTERED function or the NOT ENTERED function. For an alert box, *field* must be a LOGICAL variable. It sets the default button and returns the user's choice. If the alert box has two buttons, they represent the values TRUE and FALSE, respectively. If the alert box has three buttons, they represent the values TRUE, FALSE, and the Unknown value (?), respectively.

AS *datatype*

Defines *field* as a variable of type *datatype*. You must use this option or the LIKE option if *field* has not been previously defined.

LIKE *field*

Defines the field specified in SET or UPDATE as a database field or a previously defined variable.

FORMAT *string*

The format that you want to use to display the *field* used in the SET or UPDATE option. For more information on display formats, see *OpenEdge Development: Progress 4GL Handbook*. If you do not use the FORMAT option, Progress uses the defaults shown in Table 37.

Table 37: Default display formats

Type of expression	Default format
Field	Format from schema.
Variable	Format from variable definition.
Constant character	Length of character string.
Other	Default format for the data type of the expression.

Table 38 shows the default formats for the Other expression.

Table 38: Default data type display formats

Data type	Default display format
CHARACTER	x(8)
CLASS[3]	>>>>>>9
DATE	99/99/99
DATETIME	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	->>,>>9.99
HANDLE[2]	>>>>>>9
INTEGER	->,>>>,>>9
LOGICAL	yes/no
MEMPTR[1]	See the footnote at the end of the table.
RAW[1]	See the footnote at the end of the table.
RECID	>>>>>>9
ROWID[1]	See the footnote at the end of the table.
WIDGET-HANDLE[2]	>>>>>>9

[1] You cannot display a MEMPTR, RAW, or ROWID value directly. However, you can convert it to a character string representation using the STRING function and display the result. A ROWID value converts to a hexadecimal string, "0x*hexdigits*," where *hexdigits* is any number of characters "0" through "9" and "A" through "F". A MEMPTR or RAW value converts to decimal integer string.

[2] To display a HANDLE or WIDGET-HANDLE, you must first convert it using the INTEGER function and display the result.

[3] To display a CLASS, you must first convert it using the INTEGER or STRING function and display the result.

`AUTO-RETURN`

Performs a carriage return when the field that is SET or UPDATEd is full.

`IN WINDOW` *`window`*

Specifies the window in which the message is displayed.

Examples

In this procedure, if you enter the number of a customer that does not exist, the procedure displays a message telling you the customer does not exist. If the customer does exist, the procedure displays the name and sales-rep of the customer.

r-msg.p

```
REPEAT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING cust-num NO-ERROR.
    IF NOT AVAILABLE customer
    THEN DO:
        MESSAGE "Customer with cust-num " INPUT cust-num
                " does not exist.  Please try another".
        UNDO, RETRY.
    END.
    ELSE DO:
        DISPLAY name sales-rep.
    END.
END.
```

The following example uses two alert boxes:

r-altbox.p

```
DEFINE VARIABLE cust-list AS CHARACTER VIEW-AS SELECTION-LIST
                          SINGLE SIZE 50 BY 10 LABEL "Customers".
DEFINE VARIABLE ok-status AS LOGICAL.

FORM
  cust-list
  WITH FRAME sel-frame.

ON DEFAULT-ACTION OF cust-list
   DO:
      MESSAGE "You have chosen to delete" cust-list:SCREEN-VALUE + "."
              SKIP(1)
              "Do you really want to delete this customer?"
            VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO-CANCEL
                    TITLE "" UPDATE choice AS LOGICAL.
      CASE choice:
         WHEN TRUE THEN /* Yes */
          DO:
            FIND customer WHERE name = cust-list:SCREEN-VALUE
                 EXCLUSIVE-LOCK.
            DELETE customer.
          END.
         WHEN FALSE THEN /* No */
          DO:
             MESSAGE "Deletion canceled."
                     VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
             RETURN NO-APPLY.
          END.
         OTHERWISE /* Cancel */
             STOP.
         END CASE.
   END.

FOR EACH customer BY name:
   ok-status = cust-list:ADD-LAST(customer.name).
END.

ENABLE cust-list WITH FRAME sel-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

In `r-altbox.p`, each time you select an item from the selection list, the procedure displays an alert box to ask if you want to delete the customer. If you choose the No button, then another alert box informs you that the record was not deleted.

Notes

- The MESSAGE statement always sends messages to the current output destination. If the INPUT source is the terminal, Progress displays messages in the window and also sends them to the current output destination. Compiler error messages also follow this convention.
- If you don't want messages sent to the current output destination, redirect the output to a named stream. Progress never writes messages to a named stream.

  If you want to send output to a destination other than the terminal, and you do not want messages to appear on the terminal (and if you are not using the terminal as an input source), use one of the statements in Table 39.

Table 39: Suppressing messages to the terminal

Operating system	Input from
UNIX	INPUT FROM /dev/null
Windows	INPUT FROM NUL

  Be sure to use the INPUT CLOSE statement to close the input source.
- Progress automatically clears messages after any user interaction, such as a SET, UPDATE, or PAUSE statement, but not after a READKEY statement.
- In Microsoft Windows, the message text in VIEW-AS ALERT-BOX is limited to 511 bytes. If the text is longer than 511 bytes, it is truncated.

- When you use the MESSAGE SET or MESSAGE UPDATE statement to update a field, Progress does not process any validation criteria defined for that field in the database. For example, if the validation criteria for the customer.name field is as follows:

```
Valexp: name BEGINS "a"
```

Use this statement:

```
MESSAGE UPDATE name
```

Progress lets you enter any data, including data that does not start with the letter a, into the name field.

Use the MESSAGE statement to display a message, but use the SET statement or UPDATE statement to let the user change the data in a frame rather than in the message area.

- If you are displaying a message to the message line and the combination of the text and field you name in a MESSAGE UPDATE statement exceeds the length of the message line, Progress truncates the text to fit on the message line.

```
DEFINE VARIABLE myvar AS CHARACTER FORMAT "x(60)".
MESSAGE "abcdefghijklmnopqrstuvwxyz" UPDATE myvar.
```

Here, the combination of the message text and the myvar variable exceeds 80 characters, so Progress truncates the message text.

- Using the MESSAGE statement to display decimal values results in truncating the nonsignificant zeros to the right of the decimal point.

```
DEFINE VARIABLE amt AS DECIMAL FORMAT ">>9.99"
INITIAL 1.20.
MESSAGE "Total" amt.
```

The previous procedure displays the following message:

```
"Total 1.2"
```

Use functions such as STRING and DECIMAL to control the format of a display.

- If the APPL-ALERT-BOXES attribute of the SESSION system handle is TRUE, then all your messages are displayed in alert boxes. You can also direct all system messages to alert boxes by setting the SYSTEM-ALERT-BOXES attribute of the SESSION system handle to true. You can remove the message area for a window by setting its MESSAGE-AREA attribute to FALSE before it is realized.
- If you use the SET or UPDATE options in a graphical environment, Progress automatically displays the message as an alert box.
- By default, all text in an alert box is displayed on a single line. If you want to break lines within the text, you must explicitly insert SKIP options into the message.
- If you use the OUTPUT TO statement to divert Progress error and warning messages to an output stream, Progress also diverts messages from the MESSAGE statement the same way. For more information, see the OUTPUT TO statement reference entry in this book.
- You can use the MESSAGE statement with an object reference for a class object instance. In this case, the MESSAGE statement implicitly calls the ToString( ) method of the class to convert the specified object reference to a character value before it displays the result.
- For SpeedScript, the only valid options are: *expression* and SKIP.

See also COLOR phrase, DECIMAL function, Format phrase, INTEGER function, MESSAGE-LINES function, STRING function

MESSAGE-LINES function

Returns the number of lines in the message area at the bottom of the window.

Note: Does not apply to SpeedScript programming.

Syntax

```
MESSAGE-LINES
```

Example

The following example displays a message on each available message line:

r-messl.p

```
DEFINE VARIABLE i AS INTEGER.

DO i = 1 TO MESSAGE-LINES:
   MESSAGE "This is message line" i.
END.
```

METHOD statement

Defines a method in a class, or declares a method prototype in an interface.

Note: This statement is applicable only when used in a class definition (`.cls`) file.

Syntax

```
METHOD { method-modifiers } { VOID | return-type } method-name
  ( [ parameter [, parameter ] ... ] ):

  method-body
```

method-modifiers

A set of options that specify method modifiers for this method. Method modifiers define the behavioral characteristics of the method. You can specify method modifiers in any order using the following syntax:

```
{ PRIVATE | PROTECTED | PUBLIC } [ OVERRIDE ] [ FINAL ]
```

`{ PRIVATE | PROTECTED | PUBLIC }`

Specifies the access mode for this method.

PRIVATE methods can be accessed only by the defining class. PROTECTED methods can be accessed by the defining class and any of its inheriting classes. PUBLIC methods can be accessed by the defining class, any of its inheriting classes, and any class or procedure that instantiates that class (that is, through an object reference).

When declaring a method prototype in an interface, the access mode for this method must be PUBLIC.

`OVERRIDE`

Indicates this method overrides the behavior of a method defined in an inherited super class. The method that overrides the definition of a method in an inherited super class must match with respect to the name, return type, and access mode of the method, and with respect to the number, type, and mode of the parameters.

This modifier is not valid when declaring a method prototype in an interface.

Note: To expose a method defined as PROTECTED in a super class, you can override the method by defining a method in a subclass with a PUBLIC access modifier.

For more information about overriding methods, see *OpenEdge Getting Started: Object-oriented Programming*.

`FINAL`

Indicates this method cannot be overridden by a method defined in an inheriting subclass.

This modifier is not valid when declaring a method prototype in an interface.

`VOID`

Indicates this method does not return a value.

`return-type`

The data type of the value this method returns. Progress provides the following return value data types: CHARACTER, CLASS, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INTEGER, LOGICAL, LONGCHAR, MEMPTR, RAW, RECID, ROWID and WIDGET-HANDLE.

To specify a class or interface object reference as a return value for a method, use the following syntax:

```
[ CLASS ] { type-name }
```

Progress passes the object reference associated with the class or interface (by value), not the class or interface itself.

type-name

A character string that specifies the type name of the class or interface. Specify a type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

If the specified class or interface type name conflicts with an abbreviation of a built-in Progress data type name, such as INT for INTEGER, you must specify the CLASS keyword.

method-name

The method name. The method name must be unique within the defining class and any class in its inherited class hierarchy, unless this method overrides a method in a super class (in which case, the name must be the same as the overridden method in the super class and you must specify the OVERRIDE method modifier).

(*parameter* [, *parameter*] ...)

Defines one or more parameters of the method.

For the parameter definition syntax, see the Parameter definition syntax reference entry in this book.

method-body

The body of the method definition. Define the method body using the following syntax:

```
      .
      .
      .
method-logic
      .
      .
      .
END [ METHOD ].
```

method-logic

The logic of the method, which can contain any Progress 4GL statements currently allowed within a PROCEDURE block including class-related statements, but excluding the RETURN ERROR statement. If the method returns a value, the method's logic must not reference, either directly or indirectly, statements that block I/O (namely, the CHOOSE, INSERT, PROMPT-FOR, READKEY, SET, UPDATE, and WAIT-FOR statements).

END [METHOD]

Specifies the end of the method body definition. You must end the method body definition with the END statement.

Note: When declaring a method prototype in an interface, you do not specify the method's logic or the END statement. For more information about declaring method prototypes in an interface, see the INTERFACE statement reference entry in this book.

Example

The following example shows the definition of a method in a class (which might implement a method prototype declared in an interface, as depicted in the second example):

```
METHOD PUBLIC CHARACTER GetCustomerName(INPUT inCustNum AS INTEGER):

  FIND ttCust WHERE ttCust.CustNum = inCustNum NO-ERROR.
  IF AVAILABLE ttCust THEN
    RETURN ttCust.CustName.
  ELSE
    RETURN ?.

END METHOD.
```

The following example shows the definition of a method prototype declaration in an interface (which might be implemented by a method definition in a class, as depicted in the first example):

```
INTERFACE acme.myObjs.Interfaces.ICustObj:

  METHOD PUBLIC CHARACTER GetCustomerName(INPUT inCustNum AS INTEGER).

END INTERFACE.
```

Notes

- You can terminate a METHOD statement with either a period (.) or a colon (:).
- A complete method definition must begin with the METHOD statement and end with the END statement.
- A method can access any data members in its defining class including PROTECTED and PUBLIC data members defined anywhere in its inherited class hierarchy.
- A method in a class can run another method in a class, as well as an internal or external procedure or a user-defined function in a persistent procedure.
- Local variables defined within a method scope to the end of the method definition. The value of local variables do not persist across method invocations; they are re-initialized each time you invoke the method. However, if you define a local variable within a method using the same name as a data member within the class hierarchy, the local variable takes precedence over the data member for the duration of the method.
- You cannot specify the PUBLIC, PRIVATE, or PROTECTED access modes in variable definitions in a method.
- You cannot define shared objects, work tables, temporary tables, or ProDataSet objects within the body of a method.
- Progress implements scalar and array parameters of methods as NO-UNDO variables.
- If the method results in an error, consider using an output parameter to return the error condition to the calling procedure. For more information, see *OpenEdge Getting Started: Object-oriented Programming*.

See also CLASS statement, FUNCTION statement, INTERFACE statement

MINIMUM function

Compares two or more values and returns the smallest.

Syntax

```
MINIMUM ( expression , expression [ , expression ] ... )
```

expression

A constant, field name, variable name, or expression. If there is a mixture of decimal and integer data types, decimal type is returned.

Example

This procedure prompts the user for an item number and how many of the item they want. If the number of items a user wants (stored in the want variable) is the minimum of the want variable and the on-hand field, the procedure displays an “enough in stock” message. Otherwise, the procedure displays a “not enough in stock” message.

r-minmum.p

```
DEFINE VARIABLE want LIKE on-hand LABEL "How many do you want?".
DEFINE VARIABLE ans AS LOGICAL.

REPEAT:
    PROMPT-FOR item.item-num want.
    FIND item USING item-num.
    ans = no.
    IF MINIMUM(INPUT want,on-hand) = INPUT want
    THEN DO:
        MESSAGE "We have enough" item-name "in stock.".
        MESSAGE "Any other items to check?" UPDATE ans.
        IF NOT ans THEN LEAVE.
    END.
    ELSE DO:
        MESSAGE "We only have" on-hand item-name "in stock.".
        MESSAGE "Any other items to check?"
        UPDATE ans.
        IF NOT ans THEN LEAVE.
   END.
END.
```

Notes

- When comparing character values, if at least one of the character fields is defined as case sensitive, then MINIMUM treats all of the values as case sensitive for the sake of the comparisons. If none of the values is case sensitive, MINIMUM treats lowercase letters as if they were uppercase letters.
- You can use MINIMUM to compare DATE, DATETIME, and DATETIME-TZ data. The data type that contains less information (that is, a DATE value contains less information than a DATETIME value, and a DATETIME value contains less information than a DATETIME-TZ value) is converted to the data type with more information by setting the time value to midnight, and the time zone value to the session's time zone (when the data type does not contain the time or time zone). Comparisons with DATETIME-TZ data are based on Coordinated Universal Time (UTC) date and time.

See also MAXIMUM function

MODULO operator

Determines the remainder after division.

Syntax

```
expression MODULO base
```

expression

An integer expression.

base

A positive integer expression that is the modulo base. For example, angles measured in degrees use a base of 360 for modulo arithmetic. 372 MODULO 360 is 12.

Example

This procedure determines the number of trucks required to ship a given quantity of material, and how much material is left over from a less than full truck load:

r-modulo.p

```
REPEAT:
  SET qty-avail AS INTEGER LABEL "Qty. Avail.".
  SET std-cap AS INTEGER
  LABEL "Std. Truck Capacity".
  DISPLAY TRUNCATE(qty-avail / std-cap,0)
    FORMAT ">,>>9" LABEL "# Full Loads"
    qty-avail MODULO std-cap LABEL "Qty. Left".
END.
```

Note

The *expression* must be greater than 0 for MODULO to return a correct value.

MONTH function

Evaluates a date expression and returns a month integer value from 1 to 12, inclusive.

Syntax

```
MONTH ( date )
```

```
MONTH ( datetime-expression )
```

date

A date expression where you want a month value.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The MONTH function returns the month of the date part of the DATETIME or DATETIME-TZ value.

Example

This procedure displays all the orders that have a promise-date in a month that has passed, and whose ship-date field is the Unknown value (?), which is the initial value of the ship-date field:

r-mon.p

```
FOR EACH order:
    IF (MONTH(promise-date) < MONTH(TODAY) OR
       YEAR(promise-date) < YEAR(TODAY)) AND ship-date = ?
    THEN DISPLAY order-num LABEL "Order Num" po LABEL "P.O. Num"
                 promise-date LABEL "Promised By"
                 order-date LABEL "Ordered" terms
                 WITH TITLE "These orders are overdue".
END.
```

See also

DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, ISO-DATE function, MTIME function, NOW function, TIME function, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

MTIME function

Returns an integer representing the time in milliseconds. If the MTIME function has no arguments, it returns the current number of milliseconds since midnight (similar to TIME, which returns seconds since midnight).

Syntax

```
MTIME ( [ datetime-expression ] )
```

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The MTIME function returns the time portion of *datetime-expression* in milliseconds.

If *datetime-expression* is a DATETIME-TZ, the MTIME function returns the local time relative to the time zone of the DATETIME-TZ value. For Example, a DATETIME-TZ field, *fdt*, is created in London (time zone UTC+00:00) with a value of `May 5, 2002 at 7:15:03.002 am`. MTIME(*fdt*) returns `26,103,002`, regardless of the session's time zone.

The MTIME function gets the current system time of the client or server machine that serves as the time source for applications running during the OpenEdge session (specified by the TIME-SOURCE attribute).

See also

DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, ISO-DATE function, MONTH function, NOW function, TIME function, TIME-SOURCE attribute, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

NE or <> operator

Compares two expressions and returns a TRUE value if they are not equal.

Syntax

```
expression { NE | <> } expression
```

expression

A constant, field name, variable name, or expression. The expressions on either side of the NE or must be of the same data type.

Example

This procedure displays information for all items that appear in the catalog. (The cat-page field is not equal to the Unknown value (?) or 0).

r-ne.p

```
FOR EACH item WHERE cat-page <> ? AND cat-page <> 0:
     DISPLAY item-num item-name cat-page
       WITH TITLE "Catalog Items" USE-TEXT.
END.
```

Notes

- By default, Progress uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).
- If one of the expressions has the Unknown value (?) and the other does not, the result is TRUE. If both have the Unknown value (?), the result is FALSE. For SQL, however, if one or both expressions have the Unknown value (?), then the result is the Unknown value (?).

- You can compare character strings with NE. Most character comparisons are case insensitive in Progress. That is, all characters are converted to uppercase prior to comparisons. However, it is possible to define fields and variables as case sensitive (although it is not advised, unless strict ANSI SQL adherence is required). If either *expression* is a field or variable defined as case sensitive, the comparison is case sensitive and “Smith” does not equal “smith”.

- Characters are converted to their sort code values for comparison. Using the default case-sensitive collation table, all uppercase letters sort before all lowercase letters (for example, a is greater than Z, but less than b.) Note also that in character code uppercase A is less than [, \ , ^ , _, and ’ , but lowercase a is greater than these.

- You can use NE to compare DATE, DATETIME, and DATETIME-TZ data. The data type that contains less information (that is, a DATE value contains less information than a DATETIME value, and a DATETIME value contains less information than a DATETIME-TZ value) is converted to the data type with more information by setting the time value to midnight, and the time zone value to the session's time zone (when the data type does not contain the time or time zone). Comparisons with DATETIME-TZ data are based on Coordinated Universal Time (UTC) date and time.

- You can use NE to compare one BLOB field to another. Progress performs a byte-by-byte comparison.

- You can use NE to compare a LONGCHAR variable to another LONGCHAR or CHARACTER variable. The variable values are converted to `-cpinternal` for comparison and must convert without error, or Progress raises a run-time error.

- You can use NE to compare a CLOB field only to the Unknown value (`?`).

NEW function

Checks a record buffer and returns a TRUE value if the record in that buffer is newly created. If the record was read from the database, NEW returns a FALSE value.

Syntax

```
NEW record
```

record

The name of the record buffer you want to check with the NEW function.

To use the NEW function with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

Example

This procedure enters new orders, optionally creating a customer record if one does not exist. The NEW function is later used to select alternate processing depending if a customer is newly created or already exists.

r-new.p

```
REPEAT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING cust-num NO-ERROR.
    IF NOT AVAILABLE customer
    THEN DO WITH FRAME newcus:
        MESSAGE "Creating new customer".
        CREATE customer.
        ASSIGN cust-num.
        UPDATE name address city st country.
    END.
    CREATE order.
    order.cust-num = customer.cust-num.
    IF NEW customer THEN DO:
        UPDATE order.order-num promise-date.
        order.terms = "COD".
        DISPLAY order.terms.
    END.
    ELSE UPDATE order.order-num promise-date order.terms.
END.
```

Note

The NEW function returns a TRUE value only during the transaction in which the record is created. If the scope of the record is greater than the transaction in which the record is created, the NEW function returns a FALSE value outside the transaction.

See also

AVAILABLE function, FIND statement, LOCKED function, Record phrase

NEW statement

Creates an object instance of a class, and assigns an object reference for the instance to a variable. You access a class object instance, as well as its data members and methods, using its associated object reference variable.

Note: The application is responsible for deleting the class object instance using the DELETE OBJECT statement. If the variable that contains the object reference for the instance goes out of scope before you delete the object instance, the instance remains in memory without an object reference.

Syntax\

```
object-reference = NEW type-name ( [ parameter [, parameter ] ... ] )
[ NO-ERROR ].
```

object-reference

The name of the variable to contain the object reference for the new class object instance. You use this object reference variable to access the class object instance, and its PUBLIC data members and methods.

To define an object reference variable for a class object instance, use the DEFINE VARIABLE statement with the CLASS option. For more information, see the DEFINE VARIABLE statement reference entry in this book.

type-name

A character string that specifies the type name of the class. Specify a class type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

The *class-name* must specify one of the following:

- The same class that was specified in the object reference variable definition.
- A subclass of the class specified in the object reference variable definition.
- A class that implements the interface specified in the object reference variable definition.

(*parameter* [, *parameter*] ...)

Specifies one or more parameters to pass to the constructor method defined for the class. You must provide the parameters identified by the constructor method, and the parameters must match with respect to the number, data type, and mode.

For the parameter passing syntax, see the Parameter passing syntax reference entry in this book.

For information about defining a constructor method for a class, see the CONSTRUCTOR statement reference entry in this book.

NO-ERROR

Suppresses any errors that occur during the invocation of the constructor for the class, or any class in the inherited class hierarchy, while attempting to create the class object instance or its inherited class hierarchy. For example:

- The class definition file for the class, a super class, or an interface could not be found.
- The class definition file for the class, a super class, or an interface could not be compiled.
- The run-time parameters of the constructor for the class, or a constructor for a class in the inherited class hierarchy, are not compatible.

When Progress encounters one of these errors, and the constructor method cannot create the class object instance or its inherited class hierarchy, Progress raises the ERROR condition on the NEW statement and sets the object reference to the Unknown value (?).

After the NEW statement completes, you can check the ERROR-STATUS system handle for information about errors that have occurred. For more information, see *OpenEdge Getting Started: Object-oriented Programming*.

Example

The following example shows the definition of a variable to contain the object reference for a new class object instance:

```
DEFINE VARIABLE myCustObj AS CLASS acme.myObjs.CustObj NO-UNDO.

myCustObj = NEW acme.myObjs.CustObj ( ).
```

Notes

- When you create an object instance of a class, Progress invokes the constructor for the class, and the constructor for each class in its inherited class hierarchy. At this time, the object instance gets its own copy of the data members defined in the class and all classes in its inherited class hierarchy. In addition, the object instance is added to the list of valid object instances referenced by the FIRST-OBJECT and LAST-OBJECT attributes of the SESSION system handle.

- To access PUBLIC data members, qualify the data member names with their associated object reference using the following syntax:

```
object-reference:data-member-name
```

  You cannot access PRIVATE or PROTECTED data members using an object reference. If you want to make these data members available, you must provide PUBLIC methods to do so.

- To access PUBLIC methods, qualify the method names with their associated object reference using the following syntax:

```
object-reference:method-name( [ parameter [, parameter ] ... ] ).
```

  You cannot access PRIVATE or PROTECTED methods using an object reference.

- You can use an object reference for a class object instance as a parameter or return type for methods, internal and external procedures, and user-defined functions. You can also assign an object reference to a temporary table field defined as a Progress.Lang.Object; but you cannot assign an object reference to a field in a database table. For more information, see *OpenEdge Getting Started: Object-oriented Programming*.

- You cannot pass an object reference as a parameter to an AppServer; nor can you pass a temporary table containing an object reference field as a parameter to an AppServer.

- A class object instance is not associated with a handle. Thus, you cannot use a class in any statement or function that returns a value of type HANDLE, and you cannot pass a procedure handle to a method expecting an object reference.

- You can assign one object reference variable to another object reference variable when the destination object reference (on the left side of the assignment) is defined for the same class, a super class, or an interface of the object reference being assigned (on the right side of the assignment). The destination object reference retains its class type. When you assign an object reference for a super class defined higher in the class hierarchy (on the right side of the assignment) to an object reference for a subclass defined lower in the class hierarchy (on the left side of the assignment), you must cast the object reference being assigned by using the CAST function. In any case, the assignment results in a copy of the object reference, which points to the same object instance, not a copy of the associated object instance. For more information about the CAST function, see the CAST function reference entry in this book.

- You can compare two object references for equality using the EQ or = operator, which determines if two object references are referencing the same object instance. Two object references can be equal even if you defined them for different classes in the class hierarchy. Two object references are also equal if they are both the Unknown value (?).

- You can use the Equals( ) method in the Progress.Lang.Object class to compare the data members of two object references, as long as this class provides an implementation of the Equals( ) method.

See also CAST function, CLASS statement, CONSTRUCTOR statement, DEFINE VARIABLE statement, FIRST-OBJECT attribute, FUNCTION statement, LAST-OBJECT attribute, METHOD statement

NEXT statement

Goes directly to the END of an iterating block and starts the next iteration of the block.

Syntax

```
NEXT [ label ]
```

label

The name of the block for which you want to start the next iteration. If you do not name a block, Progress starts the next iteration of the innermost iterating block that contains the NEXT statement.

Example

The FOR EACH block in this procedure reads a single customer record on each iteration of the block. If the sales-rep field of a customer record does not match the sales-rep value supplied to the PROMPT-FOR statement, the NEXT statement causes Progress to do the next iteration of the FOR EACH block, bypassing the DISPLAY statement.

r-next.p

```
PROMPT-FOR customer.sales-rep LABEL "Enter salesman initials"
    WITH SIDE-LABELS CENTERED.
FOR EACH customer:
    IF sales-rep <> INPUT sales-rep
    THEN NEXT.
    DISPLAY cust-num name city state WITH CENTERED USE-TEXT.
END.
```

See also

LEAVE statement

NEXT-PROMPT statement

Specifies the field in which you want to position the cursor during the next input operation that involves that field in a frame.

Note: Does not apply to SpeedScript programming.

Syntax

```
NEXT-PROMPT field [ frame-phrase ]
```

field

Indicates the name of the input field in which you want to place the cursor the next time the user supplies input to the frame. If the field you name is not an input field in the frame, Progress disregards the NEXT-PROMPT statement.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see theFrame phrase reference entry.

Example

This procedure lets you update customer information. If you do not enter a value for contact, Progress positions the cursor in the contact field when the UPDATE statement is processed following the UNDO, RETRY of the FOR EACH block.

r-nprmpt.p

```
FOR EACH customer:
    UPDATE customer WITH 2 COLUMNS.
    IF contact EQ " "
    THEN DO:
       MESSAGE "You must enter a contact".
       NEXT-PROMPT contact.
       UNDO, RETRY.
    END.
END.
```

Notes

- NEXT-PROMPT is useful in an EDITING phrase because it can dynamically reposition the cursor depending on input from the user.
- When you have to do complex field checking that you are unable to do in a Dictionary validation expression or in a VALIDATE option of the Frame phrase, use NEXT-PROMPT to position the cursor after detecting an error.
- If the next data entry statement involving the frame specified with NEXT-PROMPT does not use the indicated NEXT-PROMPT field, then Progress ignores the NEXT-PROMPT statement.
- The NEXT-PROMPT statement can affect default frame layout. In this procedure, Progress prompts for a and b (in that order):

r-nextp.p

```
DEFINE VARIABLE a AS CHARACTER.
DEFINE VARIABLE b AS CHARACTER.
UPDATE a b.
```

However, if you include NEXT-PROMPT b before the update statement, as shown in the following procedure, Progress prompts for b first and a second.

r-nextp1.p

```
DEFINE VARIABLE a AS CHARACTER.
DEFINE VARIABLE b AS CHARACTER.
NEXT-PROMPT b.
UPDATE a b.
```

See also EDITING phrase, Frame phrase

NEXT-VALUE function

Returns the next integer value of a static sequence, incremented by the positive or negative value defined in the Data Dictionary.

Syntax

```
NEXT-VALUE ( sequence [ , logical-dbname ] )
```

sequence

An identifier that specifies the name of a sequence defined in the Data Dictionary.

logical-dbname

An identifier that specifies the logical name of the database in which the sequence is defined. The database must be connected. If multiple databases are connected, you can omit this parameter if you specify a sequence that is unique to one of the databases.

Example

The following trigger procedure uses the Next-Item-Num sequence to set the item-num field for a new item record:

r-critem.p

```
TRIGGER PROCEDURE FOR Create OF Item.

/* Automatically assign a unique item number using Next-Item-Num seq */

ASSIGN Item.Item-Num = NEXT-VALUE(Next-Item-Num).
```

Notes

- If *sequence* is a cycling sequence, and the NEXT-VALUE function increments the sequence beyond its upper limit (for positive increments) or decrements the sequence beyond its lower limit (for negative increments), the function sets and returns the initial value defined for the sequence.
- If *sequence* is a terminating sequence, and the NEXT-VALUE function attempts to increment the sequence beyond its upper limit (for positive increments) or decrement the sequence beyond its lower limit (for negative increments), the function returns the Unknown value (?) and leaves the current sequence value unchanged. Once a sequence terminates, NEXT-VALUE continues to return the Unknown value (?) for the specified sequence until it is reset to a new value with the CURRENT-VALUE statement, or its definition is changed to a cycling sequence. After changing the sequence definition to cycle, the first use of NEXT-VALUE for the sequence sets and returns its initial value.
- The value of a sequence set by the NEXT-VALUE function persists in the database until the next CURRENT-VALUE statement or NEXT-VALUE function is invoked for the sequence, or until the sequence is deleted from the database.
- You cannot invoke the NEXT-VALUE function from within a WHERE clause. Doing so generates a compiler error because the value returned by the NEXT-VALUE function can result in ambiguous expressions. To use a result from the NEXT-VALUE function in a WHERE clause, assign the result to a variable and use the variable in the WHERE clause instead.
- You can use any combination of the NEXT-VALUE function, CURRENT-VALUE function, CURRENT-VALUE statement, and their dynamic versions. Use the dynamic version when you don’t know what the database name or sequence name is at runtime.

See also

CURRENT-VALUE function, CURRENT-VALUE statement, DYNAMIC-CURRENT-VALUE function, DYNAMIC-CURRENT-VALUE statement, DYNAMIC-NEXT-VALUE function, NEXT-VALUE function

NORMALIZE function

Normalizes a character string based on the specified Unicode normalization form.

Syntax

```
NORMALIZE ( string, normalization-form )
```

string

The source string to normalize. The value may be of type CHARACTER or LONGCHAR.

If the string is a CHARACTER value, `-cpinternal` must be set to UTF-8. If the string is a LONGCHAR value, its code page can be any form of Unicode (for example, UTF-8, UTF-16, or UTF-32). This function returns a value of the same data type as the source string.

normalization-form

A character expression that evaluates to one of the following Unicode normalization forms:

- **NFD** — Canonical Decomposition.
- **NFC** — Canonical Decomposition, followed by Canonical Composition.
- **NFKD** — Compatibility Decomposition.
- **NFKC** — Compatibility Decomposition, followed by Canonical Composition.
- **NONE** — Returns the source string unchanged.

NOT operator

Returns TRUE if an expression is false, and FALSE if an expression is true.

Syntax

```
NOT expression
```

expression

A logical expression whose value is logical, that is TRUE/FALSE, YES/NO.

Example

In this procedure, if the user enters the number of a customer that does not exist, the procedure displays a message that the customer does not exist and the user must try again. If the customer does exist, the procedure displays the name and phone number of the customer.

r-not.p

```
REPEAT:
  PROMPT-FOR customer.cust-num.
  FIND customer USING cust-num NO-ERROR.
  IF NOT AVAILABLE customer
  THEN DO:
    MESSAGE "Customer with cust-num:" INPUT cust-num
      " does not exist. Please try another.".
    UNDO, RETRY.
  END.
  ELSE DO:
    DISPLAY name phone.
  END.
END.
```

See also

AND operator, OR operator

NOT ENTERED function

Returns a TRUE value if a frame field was not modified during the last INSERT, PROMPT-FOR, SET, or UPDATE statement.

Note: Does not apply to SpeedScript programming.

Syntax

```
[ FRAME frame ] field NOT ENTERED
```

FRAME *frame*

The frame name that contains the field named by the *field* argument. If you do not name a frame, the NOT ENTERED function starts with the current frame and searches outward until it finds the field you name with the *field* argument.

field

The name of the field or variable you are checking.

Example

This procedure displays the cust-num, name, and credit-limit for each customer. For each customer, the procedure prompts the user for a new credit-limit value. The NOT ENTERED function tests to see if you enter a value. If you enter a value and it is different from the present value of credit-limit, the procedure displays the old and new credit-limit values. If you enter the same value or no value, the procedure displays a message that the credit-limit has not been changed.

r-nenter.p

```
DEFINE VARIABLE new-max LIKE credit-limit.

FOR EACH CUSTOMER:
    DISPLAY cust-num name credit-limit LABEL "current max credit"
       WITH FRAME a 1 DOWN ROW 1.
    SET new-max LABEL "new max credit"
       WITH SIDE-LABELS NO-BOX ROW 10 FRAME b.
    IF new-max NOT ENTERED OR new-max = credit-limit THEN DO:
        DISPLAY "No Change In credit-limit" WITH FRAME d ROW 15.
        NEXT.
    END.
    DISPLAY "Changing Credit Limit of" name SKIP
            "from" credit-limit "to"
            new-max WITH FRAME c ROW 15 NO-LABELS.
    credit-limit = new-max.
END.
```

Note

If you use a field or variable referenced with NOT ENTERED in more than one frame, then Progress uses the value in the frame most recently introduced in the procedure. To make sure you are using the appropriate frame, use the FRAME option with the NOT ENTERED function to reference a particular frame.

See also

ENTERED function

NOW function

Returns the current system date, time, and time zone as a DATETIME-TZ.

The NOW function returns the system date and time of the client or server machine that serves as the time source for applications running during the OpenEdge session (specified by the TIME-SOURCE attribute).

Syntax

```
NOW
```

Example

Following is an example of using the NOW function:

```
DEF VAR v-datetime as DATETIME.
DEF VAR v-datetime-tz as DATETIME-TZ.

v-datetime = NOW.
v-datetime-tz = NOW.
```

See also

DATE function, DATE-FORMAT attribute, DATETIME function, DATETIME-TZ function, DAY function, ETIME function, ISO-DATE function, MONTH function, MTIME function, TIME function, TIME-SOURCE attribute, TIMEZONE function, TODAY function, WEEKDAY function, YEAR function, YEAR-OFFSET attribute

NUM-ALIASES function

Returns an integer value that represents the number of aliases defined. The NUM-ALIASES function uses no arguments.

Syntax

```
NUM-ALIASES
```

Example

This procedure displays the number of defined aliases. It also displays the aliases and logical database names of all connected databases.

r-numal.p

```
DEFINE VARIABLE I AS INTEGER.

DISPLAY NUM-ALIASES LABEL "Number of Defined Aliases:".
REPEAT I = 1 TO NUM-ALIASES.
    DISPLAY ALIAS(I) LABEL "Aliases"
            LDBNAME(ALIAS(I)) LABEL "Logical Database".
END.
```

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement,FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function, SDBNAME function

NUM-DBS function

Takes no arguments; returns the number of connected databases.

Syntax

```
NUM-DBS
```

Example

This procedure uses NUM-DBS to display the logical name and database restrictions of all connected databases:

r-numdbs.p

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i = 1 TO NUM-DBS:
  DISPLAY LDBNAME(i) DBRESTRICTIONS(i) FORMAT "x(40)".
END.
```

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, PDBNAME function, SDBNAME function

NUM-ENTRIES function

Returns the number of elements in a list of character strings.

Syntax

```
NUM-ENTRIES ( list [ , character ] )
```

list

A character expression containing a list of character strings separated with a character delimiter. The list can be a variable of type CHARACTER or LONGCHAR. NUM-ENTRIES returns the number of elements in the list. Specifically, NUM-ENTRIES returns the number of delimiters plus 1, and it returns 0 if *list* equals the empty string ("").

character

A delimiter you define for the list. The default is a comma (,). This allows functions to operate on non-comma-separated lists. If you use an alphabetic character, this delimiter is case sensitive.

Examples

This procedure uses NUM-ENTRIES and ENTRY to loop through a list of regions and display them, one per line. Since there are obviously five regions, the REPEAT statement, REPEAT i=1 TO 5, works fine here.

r-n-ent1.p

```
DEFINE VARIABLE i AS INTEGER.
DEFINE VARIABLE regions AS CHARACTER INITIAL "Northeast,
  Southest,Midwest,Northwest,Southwest".
REPEAT i=1 TO NUM-ENTRIES(regions):
  DISPLAY ENTRY(i,regions) FORMAT "x(12)".
END.
```

In the following example, PROPATH is a comma-separated list of unknown length:

r-n-ent2.p

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i=1 TO NUM-ENTRIES(PROPATH):
  DISPLAY ENTRY(i,PROPATH) FORMAT "x(64)".
END.
```

This procedure uses NUM-ENTRIES to loop through the PROPATH (a comma-separated list of directory paths) and print the directories, one per line.

This example uses a list that does not use commas as a delimiter. This procedure returns a value of 13:

r-n-ent3.p

```
DEFINE VARIABLE sentence AS CHARACTER.
  sentence  = "This sentence would be seven words long "
      + "if it were six words shorter".
DISPLAY NUM-ENTRIES(sentence," ").
```

Note The NUM-ENTRIES function is multi-byte enabled. The specified *list* can contain entries that have multi-byte characters and the *character* delimiter can be a multi-byte character.

See also ENTRY function

NUM-RESULTS function

Returns the number of rows currently in the results list of a scrolling query. The results list is initialized when the query is opened. Depending on the query, the entire list is built immediately upon opening or it is gradually as needed.

Syntax

```
NUM-RESULTS ( query-name )
```

query-name

A character expression that evaluates to the name of a currently open, scrolling query. If *query-name* does not resolve to the name of a query, or if the query is not open or not scrolling, then the function returns the Unknown value (?).

Note: Searching for a query using a handle is more efficient than a character expression. Progress resolves a character expression at runtime by searching in the current routine for a static query with that name. If not found, Progress searches the enclosing main procedure. If still not found, Progress searches up through the calling programs of the current routine, and their main procedures. Since a handle uniquely identifies a query, no such search is required. Use the query object handle's NUM-RESULTS attribute to avoid a runtime search.

Example

The following example uses the NUM-RESULTS function in a message to report on the number of rows in a browse. Note that the query is opened with the PRESELECT option so that the entire results list is built immediately. Otherwise, NUM-RESULTS might not return the total number of rows in the browse. When you run this procedure and choose a button, Progress selects certain rows within the browse and then reports on the number of rows selected and the total number of rows in the browse.

r-brownr.p *(1 of 2)*

```
DEFINE VARIABLE curr-rec AS ROWID.
DEFINE VARIABLE status-ok AS LOGICAL.
DEFINE VARIABLE threshold LIKE customer.credit-limit INITIAL 25000.

DEFINE BUTTON no-orders-custs LABEL "No Orders".
DEFINE BUTTON hi-cred-custs LABEL "High Credit".

DEFINE QUERY qry FOR customer.
DEFINE BROWSE brws QUERY qry DISPLAY cust-num name country credit-limit
   WITH 10 DOWN MULTIPLE.

FORM
   brws SKIP(1)
   no-orders-custs hi-cred-custs
   WITH FRAME brws-frame.

FORM
   threshold
   WITH FRAME thresh-frame VIEW-AS DIALOG-BOX
        TITLE "Set Threshold" SIDE-LABELS.

ON CHOOSE OF no-orders-custs DO:
   /* Select those customers with no orders. */

   status-ok = brws:DESELECT-ROWS().
   HIDE MESSAGE.
   FOR EACH customer NO-LOCK WHERE NOT CAN-FIND(FIRST order OF customer):
      /* Position query to this record and then select row in browse. */
      curr-rec = ROWID(customer).
      REPOSITION qry TO ROWID curr-rec.
      status-ok = brws:SELECT-FOCUSED-ROW().
      IF NOT status-ok
      THEN MESSAGE "Could not select row.".
   END.

   /* Report number of selected rows and position to first selected. */
  MESSAGE brws:NUM-SELECTED-ROWS "of" NUM-RESULTS("qry")
          "rows have been selected.".

   IF brws:NUM-SELECTED-ROWS > 0
   THEN status-ok = brws:SCROLL-TO-SELECTED-ROW(1).
END.
```

r-brownr.p *(2 of 2)*

```
ON CHOOSE OF hi-cred-custs DO:
   /* Select customers with high credit limits. */

   status-ok = brws:DESELECT-ROWS().
   HIDE MESSAGE.

   /* Get credit-limit threshold value. */
   UPDATE threshold WITH FRAME thresh-frame.
   FOR EACH customer NO-LOCK WHERE customer.credit-limit >= threshold:
      /* Position query to this record and then select row in browse. */
      curr-rec = ROWID(customer).
      REPOSITION qry TO ROWID curr-rec.
      status-ok = brws:SELECT-FOCUSED-ROW().
      IF NOT status-ok
      THEN MESSAGE "Could not select row.".
   END.

   /* Report number of selected rows and position to first selected. */
   MESSAGE brws:NUM-SELECTED-ROWS "of" NUM-RESULTS("qry")
           "rows have been selected.".

   IF brws:NUM-SELECTED-ROWS > 0
   THEN status-ok = brws:SCROLL-TO-SELECTED-ROW(1).
END.

OPEN QUERY qry PRESELECT EACH customer.

ENABLE ALL WITH FRAME brws-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- To use the NUM-RESULTS function with a query, the query must be associated with a browse widget or you must define the query with the SCROLLING option. For more information on query definitions, see the reference entry for the DEFINE QUERY statement.
- If the query is empty, NUM-RESULTS returns 0.
- When possible, Progress performs optimizations for GET LAST and REPOSITION statements. These optimizations make the results list invalid. At that point, NUM-RESULTS returns the Unknown value (?). These optimizations do not occur if the query is opened with the PRESELECT option or has an associated browse widget.

See also

CLOSE QUERY statement, CURRENT-RESULT-ROW function, DEFINE BROWSE statement, DEFINE QUERY statement, GET statement, NUM-RESULTS attribute, OPEN QUERY statement, QUERY-OFF-END function, REPOSITION statement

ON ENDKEY phrase

Describes the processing that occurs when the ENDKEY condition occurs during a block. This condition usually occurs when the user presses **END-ERROR** during the first interaction of a block iteration, or any time the user presses a defined **END-KEY**.

If you use a REPEAT or FOR EACH block, the default processing for ENDKEY is to undo all the processing in the current iteration of the block, then leave the block and continue on to any remaining statements in the procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
ON ENDKEY UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN { ERROR | NO-APPLY } [ return-string ]
  ]
```

label1

The name of the block whose processing you want to undo. If you do not name a block with *label1*, ON ENDKEY UNDO undoes the processing of the block started by the statement that contains the ON ENDKEY phrase.

LEAVE [*label2*]

Indicates that, after undoing the processing of a block,

Progress leaves the block labeled *label2*. If you do not name a block, Progress leaves the block containing the ON ENDKEY phrase. After leaving a block, Progress continues on with any remaining processing in a procedure.

LEAVE is the default if you do not specify LEAVE, NEXT, RETRY, or RETURN.

NEXT [*label2*]

Indicates that, after undoing the processing of a block, Progress should execute the next iteration of the block you name with the *label2* option. If you do not name a block with the NEXT option, Progress executes the next iteration of the block labeled *label1*.

RETRY [*label1*]

Indicates that, after undoing the processing of a block, Progress should repeat the same iteration of the block that you name with the *label1* option.

RETRY is the default if you do not specify of LEAVE, NEXT, RETRY, or RETURN.

RETURN [ERROR | NO-APPLY]

Returns to the calling procedure, or if there is no calling procedure, returns to the Progress Editor. Specifying ERROR causes the ERROR condition in the calling procedure. This causes the current subtransaction to be undone. You cannot specify ERROR within a user-interface trigger block. You can specify the NO-APPLY option only within a user-interface trigger block to prevent Progress from performing the default behavior for that event. For example, the default behavior for an character key press in a fill-in field is to echo the character in the field.

return-string

If you specify *return-string*, the string you provide is passed to the calling procedure. That procedure can use the RETURN-VALUE function to read the returned value.

Example

In this procedure, if the user presses **END-ERROR** or **END-KEY** while changing the credit-limit field, any changes made during the current iteration of the block are undone, and the same iteration is run again. If this procedure did not use the ON ENDKEY phrase and the user pressed **END-ERROR**, the procedure ends because the default ENDKEY action is UNDO, LEAVE. After leaving the FOR EACH block, the procedure ends because there are no more statements.

r-endky.p

```
ON WINDOW-CLOSE OF CURRENT-WINDOW
   STOP.

FOR EACH customer ON ENDKEY UNDO, RETRY:
     DISPLAY cust-num name credit-limit.
     SET credit-limit VALIDATE(credit-limit > 0,"non-zero credit limit").
END.
```

See also

ON ERROR phrase, ON QUIT phrase, ON STOP phrase, RETURN statement, RETURN-VALUE function

ON ERROR phrase

Describes the processing that occurs when there is an error in a block. If you are using a REPEAT block or a FOR EACH block, and an error occurs, all of the processing that has been done in the current iteration of the block is undone, and Progress retries the block iteration where the error occurred. (If Progress detects that a RETRY of a FOR or iterating DO block would produce an infinite loop, it performs a NEXT instead. For more information, see *OpenEdge Development: Progress 4GL Handbook*.)

Syntax

```
ON ERROR UNDO
  [ label1 ]
  [     , LEAVE [ label2 ]
      | , NEXT [ label2 ]
      | , RETRY [ label1 ]
      | , RETURN { ERROR | NO-APPLY } [ return-string ]
  ]
```

label1

The name of the block whose processing you want to undo. If you do not name a block with *label1*, ON ERROR UNDO undoes the processing of the block started by the statement that contains the ON ERROR phrase.

LEAVE [*label2*]

Indicates that after undoing the processing of a block, Progress leaves the block labeled *label2*. If you do not name a block, Progress leaves the block labeled with *label1*.

NEXT [*label2*]

Indicates that after undoing the processing of a block, Progress executes the next iteration of the block you name with the *label2* option. If you do not name a block with the NEXT option, Progress executes the next iteration of the block labeled with *label1*.

RETRY [*label1*]

Indicates that after undoing the processing of a block, Progress repeats the same iteration of the block you name with the *label1* option.

RETRY is the default processing if you do not use LEAVE, NEXT, RETRY, or RETURN.

RETURN [ERROR | NO-APPLY]

Returns to the calling procedure, or if there is no calling procedure, to the Progress Editor. Specifying ERROR causes the ERROR condition in the calling procedure. This causes the current subtransaction to be undone. You cannot specify ERROR within a user-interface trigger block. You can specify the NO-APPLY option only within a user-interface trigger block to prevent Progress from performing the default behavior for that event. For example, the default behavior for an character code key press in a fill-in field is to echo the character in the field.

return-string

If you specify *return-string*, the string you provide is passed to the calling procedure. That procedure can use the RETURN-VALUE function to read the returned value.

Example

In r-onerr.p, if you enter a customer number and the FIND statement is unable to find a customer with that number, Progress returns an error. If an error occurs, the ON ERROR phrase tells Progress to undo anything that was done in the current iteration and start the next iteration. Thus, you see any invalid numbers you enter, and you can continue to the next customer number you want to enter.

r-onerr.p

```
REPEAT ON ERROR UNDO, NEXT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING cust-num.
    DISPLAY name address city state country.
END.
```

See also

ON ENDKEY phrase, ON QUIT phrase, ON STOP phrase, RETURN statement, RETURN-VALUE function

ON QUIT phrase

Describes the processing that occurs when a QUIT statement is executed during a block. By default, the QUIT statement saves the current transaction and then returns to the operating system or to the tool from which the procedure was invoked (such as the Procedure Editor).

Note: Does not apply to SpeedScript programming.

Syntax

```
ON QUIT
  [ UNDO [ label1 ] ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN { ERROR | NO-APPLY } [ return-string ]
  ]
```

`UNDO [ label1 ]`

Indicates that the specified block is undone. If you do not specify the UNDO option, then the current transaction is committed when the QUIT statement is executed.

`LEAVE [ label2 ]`

Indicates that after committing or undoing the transaction, Progress leaves the block labeled *label*. If you do not name a block, Progress leaves the block with the ON QUIT phrase in its heading.

`NEXT [ label2 ]`

Indicates that after committing or undoing the transaction, Progress executes the next iteration of the block you name with the *label* option. If you do not name a block with the NEXT option, Progress executes the next iteration of the block with the ON QUIT phrase in its heading.

RETRY [*label1*]

Indicates that after committing or undoing the processing of a block, Progress repeats the same iteration of the block that was undone or committed.

RETRY is the default if you do not specify LEAVE, NEXT, RETRY, or RETURN.

RETURN [ERROR | NO-APPLY]

Indicates that after undoing or committing the transaction, Progress returns to the calling procedure, or if there is no calling procedure, to the tool that invoked the procedure. Specifying ERROR causes the ERROR condition in the calling procedure. This causes the current subtransaction to be undone. You cannot specify ERROR within a user-interface trigger block. You can specify the NO-APPLY option only within a user-interface trigger block to prevent Progress from performing the default behavior for that event. For example, the default behavior for an character code key press in a fill-in field is to echo the character in the field.

return-string

If you specify *return-string*, the string you provide is passed to the calling procedure. That procedure can use the RETURN-VALUE function to read the returned value.

See also ON ENDKEY phrase, ON ERROR phrase, ON STOP phrase, QUIT statement, RETURN statement, RETURN-VALUE function

ON statement

The ON statement specifies a trigger for one or more events or redefines terminal keys for an application.

Syntax

```
ON event-list
  {     ANYWHERE
    |   { OF widget-list
            [ OR event-list OF widget-list ] ...
            [ ANYWHERE ]
        }
  }
  {     trigger-block
    |   REVERT
    |   { PERSISTENT RUN procedure
            [ ( input-parameters ) ]
        }
  }
```

```
ON event OF database-object
  [ referencing-phrase ]
  [ OVERRIDE ]
  { trigger-block | REVERT }
```

```
ON key-label key-function
```

```
ON "WEB-NOTIFY" ANYWHERE  { trigger-block }
```

event-list

A comma-separated list of user-interface events for which you want to define a trigger. If any of the specified events occurs for any of the specified widgets, the trigger executes.

For a list of valid events for each widget type, see the reference page for that widget type. For information on all user interface events, see the "Events Reference" section on page 2171.

widget-list

A comma-separated list of widgets or procedure handles to which the event is applied. See the Widget phrase reference entry for more information on referencing widgets.

If a specified event occurs for any of the specified widgets, the trigger executes. If you specify a list of widgets, all events specified must be user-interface events.

ANYWHERE

You can specify ANYWHERE either with a list of widgets or instead of a list of widgets. Without a list of widgets, ANYWHERE specifies that the trigger executes when one of the specified events occurs for any widget that does not already have a specific trigger for that event. This lets you define a default trigger for the event within the application. With a list of widgets, ANYWHERE specifies that the trigger executes when one of the specified events occurs for any specified widget or for any contained widget that does not already have a specific trigger for that event. This lets you set up a default trigger for a frame or window.

event

A database event: CREATE, DELETE, FIND, WRITE or ASSIGN. If the specified event occurs for the specified table or field, the trigger executes. For database events, you can specify only one event. For more information on these events, see *OpenEdge Development: Progress 4GL Handbook*.

database-object [*referencing-phrase*]

The name of a database table or field to which the event is applied. If you specify a *database-object*, the event specified must be a database event. You cannot specify a metaschema table or field (a table or field named with an initial underscore) as the *database-object.*

The *referencing-phrase* is valid only for WRITE and ASSIGN triggers. For WRITE triggers you can specify a name for the record before the WRITE operation and a name for the record after the WRITE operation. This allows you to reference both versions of the record within the trigger. This is the syntax for WRITE trigger:

```
NEW [ BUFFER ] new-record OLD [ BUFFER ] old-record
```

For an ASSIGN trigger, you can specify a name for the old field value. This is the syntax:

```
OLD [ VALUE ] old-field-name
```

OVERRIDE

Specifies that the database trigger you are defining overrides the schema trigger for the same event. You can override a schema trigger only if it is defined as overridable in the Data Dictionary. If you do not use the OVERRIDE option, then the session trigger executes first and then the schema trigger.

trigger-block

A trigger block is either a single 4GL statement or a set of statements grouped by DO and END statements. The trigger block is executed when one of the specified events is applied to one of the specified widgets or tables.

REVERT

If you specify this option, any non-persistent trigger defined in this procedure for the event is reverted. If a trigger had also been defined for the event in a previous procedure, that previous trigger again takes effect. Progress ignores any attempt to revert a persistent trigger.

PERSISTENT RUN *procedure* [(*input-parameters*)]

Specifies a persistent trigger; that is, a trigger that remains in effect after the current procedure terminates. Normally, a trigger remains in effect only until the procedure or trigger in which it is defined ends. You can specify a persistent trigger only for user-interface events. A persistent trigger must be a procedure specified by *procedure*. The trigger procedure can take one or more input parameters; it cannot have any output parameters. The parameters of the trigger procedure are evaluated when you define the trigger; they are **not** re-evaluated when the trigger executes.

key-label

The label of the key for which you want to define a specific action. See *OpenEdge Development: Programming Interfaces* for a list of key labels.

On UNIX, all of the special Progress keys are defined in the PROTERMCAP file supplied with Progress. If the key for which you are defining an action is not already in PROTERMCAP, you must add a definition for that key. Keys that you can name that do not require a PROTERMCAP definition are **CTRL**, **RETURN**, **BACKSPACE**, **TAB**, and **DEL**.

In Windows, keys are predefined as described in the handling user input section of *OpenEdge Development: Programming Interfaces*.

key-function

The action you want Progress to take when the user presses the key associated with *key-label*. The *key-function* value can be one of the following:

ABORT	END	LEFT-END
BACKSPACE	END-ERROR	NEXT-FRAME
BACK-TAB	ENDKEY	PREV-FRAME
BELL	ENTER-MENUBAR	RECALL
CLEAR	ERROR	RETURN
CURSOR-DOWN	GO	RIGHT-END
CURSOR-LEFT	HELP	SCROLL-MODE
CURSOR-RIGHT	HOME	STOP
CURSOR-UP	INSERT-MODE	TAB
DELETE-CHARACTER		

Examples

The following example defines a WRITE trigger for the customer table:

r-oncst.p

```
ON WRITE OF customer NEW new-cust OLD old-cust
  DO:
     IF new-cust.city <> old-cust.city AND
        new-cust.postal-code = old-cust.postal-code
     THEN DO:
         MESSAGE "Must update postal code, too.".
         RETURN ERROR.
     END.
  END.
FOR EACH customer:
   UPDATE customer.
END.
```

The trigger compares the customer record before the write with the customer record after the write. If the city has changed and the postal code has not changed, the trigger displays a message and cancels the write operation.

The following example uses the ON statement to set up a trigger for two buttons:

r-widget.p

```
DEFINE BUTTON b_next LABEL "Next".
DEFINE BUTTON b_prev LABEL "Previous".
DEFINE BUTTON b_quit LABEL "Quit".
DEFINE FRAME butt-frame
  b_next b_prev
  WITH CENTERED ROW SCREEN-LINES - 1.
DEFINE FRAME info
 customer.cust-num customer.name
  b_quit AT ROW-OF customer.cust-num + 2 COLUMN-OF customer.cust-num + 18
  WITH CENTERED TITLE "Customers"  ROW 2 1 COL.

ON CHOOSE OF b_next, b_prev
DO:
  IF SELF:LABEL = "Next" THEN
    FIND NEXT customer NO-LOCK.
  ELSE FIND PREV customer NO-LOCK.
  DISPLAY customer.cust-num customer.name WITH FRAME info.
END.

ENABLE b_next b_prev WITH FRAME butt-frame.
ENABLE b_quit WITH FRAME info.

WAIT-FOR END-ERROR OF FRAME butt-frame OR
      CHOOSE OF b_quit IN FRAME info FOCUS b_next IN FRAME butt-frame.
```

The following procedure sets up mappings for **GO**, **HELP**, and **END** and defines **CTRL+X** to ring the terminal bell:

r-onstmt.p

```
ON F1 GO.       /* F1 will now perform the GO function */
ON F2 HELP.     /* F2 will now perform the HELP function   */
ON CTRL-X BELL.  /* The Ctrl-X key will be disabled                */
ON F5 ENDKEY.  /* F5 will always raise the ENDKEY condition; never ERROR*/
```

Notes

- If you use the ON statement to redefine terminal keys, the new definitions remain in effect to the end of the session or until another ON statement changes the definition.
- A trigger defined with the ON statement remains in effect until one of the following occurs:
  - Another ON statement defines another trigger (or REVERT) for the same event and widget.
  - For a non-persistent trigger, the procedure or trigger block in which the ON statement appears terminates.
- Although each widget type responds with default system actions to a limited set of valid events, you can specify any event for any widget and execute the trigger using the APPLY statement. If the event is not a valid event for the widget type, the specified trigger executes, but no default system action occurs for the widget. You can use this feature to write triggers for procedure handles that do not otherwise respond to events.
- If *event-list* includes a MENU-DROP event for a menu or submenu, do not interact with the window manager from within the *trigger-block*. Doing so causes the window manager to lose control of the system, forcing you to reboot or restart the window manager. Actions to avoid include any window system input/output (I/O) or any lengthy processing, especially in statements that cause process interruptions, such as the PAUSE statement with or without I/O. These also include actions that can generate a warning or error message, forcing window system output. Use the NO-ERROR option on supported statements to help avoid this situation. Otherwise, check valid values, especially for run-time resources like widget handles, to prevent Progress from displaying unexpected messages.
- For SpeedScript, the only valid uses of the ON statement are specifying a trigger for a database event or for specifying a trigger for a WEB-NOTIFY event (the ON "WEB-NOTIFY" ANYWHERE syntax).

See also APPLY statement, Widget phrase

ON STOP phrase

Describes the processing that occurs when the STOP condition occurs during a block. This condition occurs when a user presses STOP, a STOP statement is executed, or certain internal conditions occur within Progress. The **STOP** key is usually mapped to **CTRL+BREAK** (Windows) or **CTRL+C** (UNIX). By default, the STOP condition undoes all active transactions and returns to the startup procedure or the Procedure Editor.

Syntax

```
ON STOP UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN { ERROR | NO-APPLY } [ return-string ]
  ]
```

label1

The name of the block whose processing you want to undo. If you do not name a block with *label1*, ON STOP UNDO undoes the processing of the block started by the statement that contains the ON STOP phrase.

LEAVE [*label2*]

Indicates that after undoing the processing of a block, Progress leaves the block labeled *label2*. If you do not name a block, Progress leaves the block labeled with *label1*.

NEXT [*label2*]

Indicates that after undoing the processing of a block, Progress executes the next iteration of the block you name with the *label2* option. If you do not name a block with the NEXT option, Progress executes the next iteration of the block labeled with *label1*.

RETRY [*label1*]

Indicates that after undoing the processing of a block, Progress repeats the same iteration of the block you name with the *label1* option.

RETRY is the default processing if you do not use LEAVE, NEXT, RETRY, or RETURN.

RETURN [ERROR | NO-APPLY]

Returns to the calling procedure, or if there is no calling procedure, returns to the Progress Editor. Specifying ERROR causes the ERROR condition in the calling procedure. This causes the current subtransaction to be undone. You cannot specify ERROR within a user-interface trigger block. You can specify the NO-APPLY option only within a user-interface trigger block to prevent Progress from performing the default behavior for that event. For example, the default behavior for an character code key press in a fill-in field is to echo the character in the field.

return-string

If you specify *return-string*, the string you provide is passed to the calling procedure. That procedure can use the RETURN-VALUE function to read the returned value.

Examples

This procedure lets you update the credit-limit field for each customer. If you enter a value greater than 100,000, the program raises the STOP condition. Since you specified an UNDO, RETRY for a STOP, the procedure starts the iteration over and allows you to enter another value.

r-ostop.p

```
FOR EACH customer ON STOP UNDO, RETRY:
   DISPLAY cust-num name credit-limit.
   UPDATE credit-limit.
   IF credit-limit > 100000
   THEN STOP.
END.
```

The ON STOP phrase is especially useful to trap the STOP condition that results when a user cancels out of a record lock conflict in an application. The `r-ostop2.p` procedure is a simple record navigation and update utility that finds `Salesrep` records with the SHARE-LOCK condition. The user can update the values of a `Salesrep` record in the frame and choose the Assign button to assign the new values to the database. If the user attempts to update a `Salesrep` record that another user already has in the SHARE-LOCK condition, the `r-ostop2.p` procedure freezes as a result of the record locking conflict. Progress displays a message asking the user to wait for the other user to relinquish the lock on the record or to press the STOP key to abort the operation.

By default, the STOP key aborts the procedure. The ON STOP phrase on the DO TRANSACTION block in the `r-ostop2.p` procedure captures the STOP condition and returns control to the procedure.

r-ostop2.p

```
DEFINE BUTTON buta LABEL "Find Next".
DEFINE BUTTON butb LABEL "Assign".
DEFINE BUTTON butc LABEL "Done".
DEFINE VARIABLE methRtn AS LOGICAL NO-UNDO.

DEFINE FRAME a
  Salesrep.Sales-rep SKIP Salesrep.Rep-Name SKIP Salesrep.Region SKIP
  Month-Quota[1] Month-Quota[7] SKIP
  Month-Quota[2] Month-Quota[8] SKIP
  Month-Quota[3] Month-Quota[9] SKIP
  Month-Quota[4] Month-Quota[10] SKIP
  Month-Quota[5] Month-Quota[11] SKIP
  Month-Quota[6] Month-Quota[12] SKIP(1)
  buta     butb    Butc
  WITH 1 DOWN NO-BOX SIDE-LABELS.

/*******TRIGGERS*******/
ON CHOOSE OF buta DO:
  FIND NEXT Salesrep SHARE-LOCK.
  IF NOT AVAILABLE(Salesrep) THEN MESSAGE "No Next Salesrep".
  DISPLAY Salesrep WITH FRAME a.
END.

ON CHOOSE OF butb DO:
  DO TRANSACTION ON STOP UNDO, LEAVE:
    ASSIGN Salesrep.Sales-rep Salesrep.Rep-Name Salesrep.Region.
  END.
END.

ON CHOOSE OF butc DO:
  APPLY "ENDKEY" TO FRAME a.
END.

/*******MAIN BLOCK*******/
FIND FIRST Salesrep SHARE-LOCK.
DISPLAY Salesrep WITH FRAME a.
ENABLE ALL WITH FRAME a.
WAIT-FOR ENDKEY OF FRAME a FOCUS buta.
```

See also ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, RETURN statement, RETURN-VALUE function, STOP statement

OPEN QUERY statement

Opens a query, which might have been previously defined in a DEFINE QUERY statement. Opening a query makes it available for use within a GET statement, or in a browse widget.

Syntax

```
OPEN QUERY query { FOR | PRESELECT } EACH record-phrase
[ , { EACH | FIRST | LAST } record-phrase ] ...
[ query-tuning-phrase ]
[ BY expression [ DESCENDING ]
| COLLATE ( string , strength [ , collation ] ) [ DESCENDING ]
] ...
[ INDEXED-REPOSITION ]
[ MAX-ROWS num-results ]
```

query

The query to open. The query name may have been defined previously in a DEFINE QUERY statement. Otherwise, the OPEN QUERY statement implicitly defines the query.

{ FOR **|** PRESELECT **}** EACH *record-phrase*

Specifies the first buffer of the query. The following is the syntax for *record-phrase*:

```
record
 [[ LEFT ]] [ OF table ]
  [ WHERE expression ]
  [ USING [ FRAME frame ] field
      [ AND [ FRAME frame ] field ] ... ]
  [ USE-INDEX index ]
  [ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
  [ NO-PREFETCH ]
```

If the query was previously defined, the buffers referenced by the *record-phrase* must be the same buffers referenced in the DEFINE QUERY statement and in the same order. For more information, see the Record phrase reference entry.

Note that the first buffer must be qualified with EACH rather than the FIRST option. That is, the OPEN QUERY statement implies the possibility of a multi-row result, whether or not only one row is returned.

If you specify PRESELECT rather than FOR, then Progress preselects the records for the query. During the preselect process, Progress applies whatever locking is specified in the OPEN QUERY statement or, if none is specified, SHARE-LOCK. It then reads the ROWID for each record into the result list. (If you do not specify PRESELECT, Progress might pass through the records anyway to presort them. In this case, Progress applies NO-LOCK to each record during this pass.)

{ EACH **|** FIRST **|** LAST **}** *record-phrase*

Specifies subsequent buffers in the query. Each subsequent buffer specifies a join with the previous buffer(s) according to the *record-phrase*. If the query was previously defined, the buffers referenced by the *record-phrase* must be the same buffers referenced in the DEFINE QUERY statement and in the same order. For more information on specifying joins in Record phrases, see the Record phrase reference entry.

query-tuning-phrase

Allows programmatic control over the execution of a DataServer query. Following is the syntax for the *query-tuning-phrase*:

```
QUERY-TUNING
 (
    [ LOOKAHEAD [ CACHE-SIZE integer ] | NO-LOOKAHEAD ]
    [ DEBUG { SQL | EXTENDED } | NO-DEBUG ]
    [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
    [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
    [ BIND-WHERE | NO-BIND-WHERE ]
    [ INDEX-HINT | NO-INDEX-HINT ]
  )
```

For more information, see your OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

BY *expression* [DESCENDING]

Specifies the order in which records are to be returned. If an index is defined with the right leading keys to satisfy the BY clause, Progress uses that index to sort the records. Otherwise, Progress must presort the records before the first fetch when you specify BY. The DESCENDING option sorts the records in descending order (not in the default ascending order).

COLLATE (*string* , *strength* [, *collation*]) [DESCENDING]

Generates the collation value of a string after applying a particular strength, and optionally, a particular collation. The DESCENDING option sorts the records in descending order (not in default ascending order).

string

A CHARACTER expression that evaluates to the string whose collation value you want to generate.

strength

A CHARACTER expression that evaluates to a Progress comparison strength or an International Components for Unicode (ICU) comparison strength.

The Progress comparison strengths include:

RAW — Generates a collation value for the string based on its binary value.

CASE-SENSITIVE — Generates a case-sensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, Progress applies the ICU TERTIARY strength.

CASE-INSENSITIVE — Generates a case-insensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, Progress applies the ICU SECONDARY strength.

CAPS — Generates a collation value for the string based on its binary value after converting any lowercase letters in the string to uppercase letters, based on the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters.

The ICU comparison strengths include:

PRIMARY — Generates a collation value for the base characters in the string.

SECONDARY — Generates a collation value for the base characters and any diacritical marks in the string.

TERTIARY — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string.

QUATERNARY — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string, and distinguishes words with and without punctuation. ICU uses this strength to distinguish between Hiragana and Katakana when applied with the ICU-JA (Japanese) collation. Otherwise, it is the same as TERTIARY.

IGNORE-SECONDARY — Generates a case-sensitive, but diacritically-insensitive, collation value for the string.

Note: Use ICU comparison strengths only with ICU collations.

collation

A CHARACTER expression that evaluates to the name of a Progress collation table or ICU collation. If *collation* does not appear, COLLATE uses the collation table of the client.

Progress reports an error and stops execution if one of the following occurs:

- *strength* does not evaluate to a valid value.
- *collation* does not evaluate to a collation table residing in the `convmap.cp` file.
- *collation* evaluates to a collation table that is not defined for the code page corresponding to the `-cpinternal` startup parameter.

INDEXED-REPOSITION

If you specify this option, Progress attempts to optimize subsequent REPOSITION TO ROWID operations on the query. This can improve the performance of REPOSITION operations that must jump over many records in a simple query. Optimization is not possible if the database is not an OpenEdge database, or sorting or preselection is performed. In these cases, the INDEXED-REPOSITION option is ignored and no error is reported.

The optimization has some side effects. When you perform a REPOSITION TO ROWID with this optimization, Progress discards the original result list and begins a new one. Therefore, scrolling forward or backward in the list might return different records from before. Also, the values of the NUM-RESULTS and CURRENT-RESULT-ROW become invalid. If the query has an associated browse, any selections in that browse are also lost. Lastly, the vertical scrollbar thumb is disabled. Because of these side-effects, use this option selectively.

MAX-ROWS *num-results*

Specifies the maximum number of records to be returned by the query. Any other records satisfying the query are ignored and no error is raised. The limit is imposed before any sorting occurs; Progress retrieves records up to the number specified and then sorts those records.

This option is valid for scrolling queries only. You can use it to prevent a long delay that might occur if a query returns many more records than you expect.

Example The following example opens a query on the customer, order, order-line, and item table:

r-opqury.p

```
DEFINE QUERY q-order FOR
    customer FIELDS (customer.cust-num customer.name customer.phone),
    order FIELDS (order.order-num order.order-date),
    order-line FIELDS (order-line.line-num order-line.price
                       order-line.qty),
    item FIELDS (item.item-num item.item-name item.cat-desc).

OPEN QUERY q-order FOR EACH customer,
    EACH order OF customer,
    EACH order-line OF order,
    EACH item OF order-line NO-LOCK.

GET FIRST q-order.

DO WHILE AVAILABLE(customer):

  DISPLAY customer.cust-num
    customer.name skip
    customer.phone skip
    order.order-num order.order-date skip
    order-line.line-num
    order-line.price order-line.qty skip
    item.item-num item.item-name skip
    item.cat-desc VIEW-AS EDITOR SIZE 50 BY 2 SCROLLBAR-VERTICAL
    WITH FRAME ord-info CENTERED SIDE-LABELS TITLE "Order Information".

  /* Allow scrolling, but not modification, of cat-desc. */
  ASSIGN item.cat-desc:READ-ONLY IN FRAME ord-info = TRUE
         item.cat-desc:SENSITIVE IN FRAME ord-info = TRUE.

  PAUSE.

  GET NEXT q-order.

END. /* DO WHILE AVAIL(customer) */
```

Note the use of field lists in the DEFINE QUERY statement. This can improve the performance of remote database queries significantly.

Notes

- If the query you reference in an OPEN QUERY statement is already open, then that query is closed and a new query is opened.
- If you use the USE-INDEX option of the Record phrase, Progress uses only that index. Records are returned in index order.
- The locking options of the OPEN QUERY statement define the default locking for records fetched by the query. You can override the default by using a locking option in the GET statement. Note, however, that in the OPEN QUERY statement you can specify a separate lock type for each buffer; in the GET statement you can specify only one lock type that applies to all buffers in a join.
- The record locking behavior specified for a query in the DEFINE BROWSE statement overrides the record locking behavior specified with the OPEN QUERY statement. The default record locking behavior of a browse widget is NO-LOCK. The default record locking behavior of a query defined with the OPEN QUERY statement is SHARE-LOCK. If you define a query and a browse widget for the query without explicitly defining record locking behavior, the query will have the NO-LOCK behavior.
- Each time you open a query associated with a browse widget, the data in the browse is refreshed.
- You cannot use the CAN-FIND function in a WHERE clause. Doing so generates a compiler error.
- If you open a query that has already been defined with multiple buffers, you must specify the buffers in the same order in the OPEN QUERY as they were specified in the DEFINE QUERY statement.
- Once the query has been opened, you cannot change the buffers that it references, even if the query is closed and re-opened. For example, a buffer, buff1, is created for the customer table in a DEFINE QUERY or OPEN QUERY for the query, qry1. The query is run and closed. You cannot now DEFINE or OPEN qry1 with buff1 for the item table. You can reuse buffers with CREATE QUERY, but you must re-run QUERY-PREPARE.

See also

CLOSE QUERY statement, CREATE QUERY statement, CURRENT-RESULT-ROW function, DEFINE BROWSE statement, DEFINE QUERY statement, GET statement, NUM-RESULTS function, QUERY-OFF-END function, QUERY-PREPARE( ) method, REPOSITION statement

OPSYS function

Identifies the operating system being used, so that a single version of a procedure can work differently under different operating systems. Returns the value of that operating system. Valid values are “UNIX” and “WIN32".

Syntax

```
OPSYS
```

Example

This procedure produces a listing of the files in your current directory. The OPSYS function determines which operating system you are running Progress on, and uses the appropriate operating system command to produce the directory listing. The example shows the possible return values.

r-opsys.p

```
IF OPSYS = "UNIX" THEN UNIX ls.
ELSE IF OPSYS = "WIN32" THEN DOS dir.
ELSE MESSAGE OPSYS "is an unsupported operating system".
```

Note

The Progress 4GL supports an override option that enables applications that need to return the value of MS-DOS for all Microsoft operating systems to do so. For example, if you do not want the value WIN32 returned when either Windows 95 or Windows NT operating systems are recognized, you can override this return value by defining the Opsys key in the Startup section of the current environment, which may be in the registry or in an initialization file. If the Opsys key is located, the OPSYS function returns the value associated with the Opsys key on all platforms.

See also

DOS statement, UNIX statement, { } Preprocessor name reference

OR operator

Returns a TRUE value if either of two logical expressions is TRUE.

Syntax

```
expression OR expression
```

expression

A logical expression (a constant, field name, variable name or expression whose value is logical, that is, TRUE/FALSE, YES/NO).

Example

This procedure lists customers who have no postal code (postal-code = "") or that have no telephone number (phone = ""). It also displays how many customers are in the list.

r-or.p

```
FOR EACH customer WHERE postal-code = "" OR phone = "":
    DISPLAY cust-num name (COUNT) city state postal-code phone.
END.
```

See also

AND operator, NOT operator

OS-APPEND statement

Executes an operating system file append command from within Progress.

Syntax

```
OS-APPEND
  { source-filename | VALUE ( expression ) }
  { target-filename | VALUE ( expression ) }
```

source-filename

The name of the source file. (If you append file A to file B, file A is the source file.) If you specify a directory, OS-APPEND generates an error.

VALUE (*expression*)

An expression that returns the name of the source file. (If you append file A to file B, file A is the source file.) *expression* can contain constants, field names, and variable names.

target-filename

The name of the target file. (If you append file A to file B, file B is the target file.)

VALUE (*expression*)

An expression that returns the name of the target file. (If you append file A to file B, file B is the target file.) *expression* can contain constants, field names, and variable names.

Example

This procedure opens a dialog box that prompts the user to choose a source file for the append. It then prompts for a name for the target file. Finally, the procedure uses the OS-APPEND statement to append the source file to the target file.

r-os-app.p

```
DEFINE VARIABLE sourcefile AS CHARACTER NO-UNDO.
DEFINE VARIABLE targetfile AS CHARACTER FORMAT "x(20)" VIEW-AS FILL-IN.
DEFINE VARIABLE OKpressed AS LOGICAL INITIAL TRUE.

Main:
REPEAT:
  SYSTEM-DIALOG GET-FILE sourcefile
    TITLE "Choose Source File For Append"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.

  IF OKpressed = FALSE THEN
    LEAVE Main.
  UPDATE targetfile WITH FRAME appendframe.
  OS-APPEND VALUE(sourcefile) VALUE(targetfile).
END.
```

Notes

- The filenames must conform to the naming conventions of the underlying operating system.
- If *target-file* names a file that does not exist or a directory, OS-APPEND becomes an OS-COPY and a copy is created in the current or specified directory. If an error occurs during the copy, Progress deletes the partial *target-file*.
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.
- If you specify the same file for the source and the target, the append fails but OS-ERROR is not set.

See also

OS-ERROR function

OS-COMMAND statement

Escapes to the current operating system and executes an operating system command.

Syntax

```
OS-COMMAND
  [ SILENT | NO-WAIT ]
  [ NO-CONSOLE ]
  [ command-token | VALUE ( expression ) ] ...
```

SILENT

After processing an operating system command, the Progress shell pauses and prompts you to press **SPACEBAR** to continue. You can use the SILENT option to eliminate this pause. Use this option only if you are sure that the program, command, or batch file does not generate any output to the screen. Cannot be used with NO-WAIT.

NO-WAIT

In a multi-tasking environment, causes Progress to immediately pass control back to next statement after the OS-COMMAND without waiting for the operating system command to terminate. Cannot be used with SILENT.

NO-CONSOLE

While processing an operating system command, Progress creates a console window. The console window may not be cleaned up after the command is executed. You can use the NO-CONSOLE option to prevent this window from being created in the first place.

command-token | VALUE (*expression*)

One or more command words and symbols that you want to pass the operating system to execute. The VALUE option generates the command tokens included in *expression*, a character string expression. The specified combination of *command-token* and VALUE(*expression*) options can form any legal combination of commands and command options permitted by the operating system.

Example

There are two principal uses for the OS-COMMAND statement: to execute a Progress utility that has the same syntax on two or more different operating systems, and to execute an operating system statement input by a user.

In both instances, the OS-COMMAND statement eliminates the need to use the OPSYS statement to determine the operating system and then use conditional logic to execute the appropriate code. The OS-COMMAND statement, therefore, makes an application more portable.

This procedure prompts the user for an operating system command and then uses the OS-COMMAND statement to execute the command:

r-os-com.p

```
DEFINE VARIABLE comm-line AS CHARACTER FORMAT "x(70)".
REPEAT:
  UPDATE comm-line.
  OS-COMMAND VALUE(comm-line).
END.
```

Notes

- If you want to run an operating system internal command, such as Windows `dir` , do not use the NO-WAIT keyword. The results are unpredictable.
- If you want to run an application that requires Windows, you must use the NO-WAIT option.
- The NO-WAIT option is unavailable in environments that are not multi-tasking.
- The OS-COMMAND statement always sets the value for the OS-ERROR function to 0, whether or not an error occurs. Thus, an operating system error is never returned for the OS-COMMAND statement.

See also

DOS statement, OPSYS function, OS-ERROR function, UNIX statement

OS-COPY statement

Executes an operating system file copy command from within Progress.

Syntax

```
OS-COPY
  { source-filename | VALUE ( expression ) }
  { target-filename | VALUE ( expression ) }
```

source-filename

The name of the original file. If you specify a directory, OS-COPY generates an error.

VALUE (*expression*)

An expression that returns the name of the original file. *Expression* can contain constants, field names, and variable names.

target-filename

The name of the new file or directory. If you specify a directory, OS-COPY gives the target file the same name as the source file.

VALUE (*expression*)

An expression that returns the name of the new file or directory. *expression* can contain constants, field names, and variable names.

Example

This procedure opens a dialog box that prompts the user to choose a file to copy. It then prompts for a name for the copy. Finally, the procedure uses the OS-COPY statement to copy the file.

r-os-cop.p

```
DEFINE VARIABLE sourcefilename AS CHARACTER NO-UNDO.
DEFINE VARIABLE copyfilename AS CHARACTER FORMAT "x(20)" VIEW-AS FILL-IN.
DEFINE VARIABLE OKpressed AS LOGICAL INITIAL TRUE.

Main:
REPEAT:
  SYSTEM-DIALOG GET-FILE sourcefilename
    TITLE  "Choose File to Copy"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.

  IF OKpressed = FALSE THEN
    LEAVE Main.
  UPDATE copyfilename WITH FRAME copyframe.
  OS-COPY VALUE(sourcefilename) VALUE(copyfilename).
END.
```

Notes

- The filenames must conform to the naming conventions of the underlying operating system.
- If *target-file* specifies an existing file, OS-COPY overwrites the existing file.
- If *target-file* has the same name as *source-file*, the copy fails, but OS-ERROR is not set.
- If the copy terminates abnormally, Progress deletes the partial *target-file*.
- Enclose filenames that refer to physical devices in double quotes (" ").
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.

See also

OS-ERROR function

OS-CREATE-DIR statement

Executes an operating system command from within Progress that creates a new directory.

Syntax

```
OS-CREATE-DIR { dirname | VALUE ( expression ) } ...
```

dirname

The name of the directory to create. If the directory already exists, no error is generated. If a file with this name exists, an error is generated. The name can be a pathname or a simple name.

If the *dirname* is not fully qualified, Progress will prepend the current working directory to the *dirname*.

VALUE (*expression*)

An expression that returns the name of the directory to create. *Expression* can contain constants, field names, and variable names.

Example

The following procedure prompts the user for the name of a directory, then creates it. If the name you give is not fully qualified, the directory is created in your current directory.

r-os-dir.p

```
DEFINE VARIABLE stat AS INTEGER.
DEFINE VARIABLE dir_name AS CHARACTER FORMAT "x(64)"
LABEL "Enter the name of the directory you want to create.".

UPDATE dir_name.
OS-CREATE-DIR VALUE(dir_name).
stat = OS-ERROR.
IF stat NE 0 THEN
   MESSAGE "Directory not created. System Error #" stat.
```

Notes

- The directory name must conform to the naming conventions of the underlying operating system.
- If a specified directory cannot be created, Progress returns an error code.
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.

See also OS-ERROR function

OS-DELETE statement

Executes an operating system file or directory delete from within Progress. Can delete one or more files, a directory, or an entire directory branch.

Syntax

```
OS-DELETE
  { filename | VALUE ( expression ) } ...
  [ RECURSIVE ]
```

filename

The name of the files or directories to delete. If you specify a directory that is not empty, you must also specify the RECURSIVE option to delete both the files contained within the directory and the directory itself.

VALUE (*expression*)

An expression that returns the name of the files or directories to delete. *expression* can contain constants, field names, and variable names.

RECURSIVE

Instructs OS-DELETE to delete all subdirectories of the directory named in *filename*, as well as the directory itself. Before a directory or subdirectory is deleted, its files are deleted.

Example

This procedure opens a dialog box that prompts the user to choose a file to delete, then uses the OS-DELETE statement to delete the file:

r-os-del.p

```
DEFINE VARIABLE filename AS CHARACTER NO-UNDO.
DEFINE VARIABLE OKpressed AS LOGICAL INITIAL TRUE.

Main:
REPEAT:
  SYSTEM-DIALOG GET-FILE filename
    TITLE     "Choose File to Delete"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.

  IF OKpressed = FALSE THEN LEAVE Main.
    ELSE OS-DELETE VALUE(filename).
END.
```

Notes

- The filenames and directory names must conform to the naming conventions of the underlying operating system.
- You cannot use wildcard characters to specify files or directories.
- If OS-DELETE encounters files or directories that are protected against deletes, it skips over them, generates an error code, but continues to delete any unprotected files and subdirectories that are specified. If several such files or directories are encountered, OS-ERROR returns information on the last error only. If a subdirectory cannot be deleted, then the named directory is not deleted.
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.

See also

OS-ERROR function

OS-DRIVES function (Windows only)

Returns a comma-separated list of available drives.

Syntax

```
OS-DRIVES
```

Example

The following procedure populates a selection list with the output of the OS-DRIVES function, and then displays the list and prompts the user to select a drive. The procedure then informs the user that subsequent writes will be to the selected drive.

r-os-drv.p

```
DEFINE VARIABLE drives AS CHARACTER
  LABEL "Select a Drive"
  VIEW-AS SELECTION-LIST INNER-CHARS 3 INNER-LINES 5.
DEFINE FRAME f
  drives.

drives:LIST-ITEMS = OS-DRIVES.
UPDATE drives WITH FRAME f.
MESSAGE “Files will be written to drive” INPUT drives:SCREEN-VALUE.
```

Note

On platforms other than Windows, OS-DRIVES compiles and executes, but returns the empty string ("").

OS-ERROR function

Returns a Progress error code that indicates whether an execution error occurred during the last OS-APPEND, OS-COPY, OS-CREATE-DIR, OS-DELETE, OS-RENAME or SAVE CACHE statement.

Syntax

```
OS-ERROR
```

Example

The following procedure prompts the user to enter a file to delete, attempts to delete the file, and then calls the OS-ERROR function to check for an execution error. If an error occurs, the procedure branches based on the error number and responds accordingly.

r-os-err.p

```
DEFINE VARIABLE err-status AS INTEGER.
DEFINE VARIABLE filename AS CHARACTER LABEL "Enter a file to delete".

UPDATE filename.
OS-DELETE filename.
err-status = OS-ERROR.

IF err-status <> 0 THEN
  CASE err-status:
  WHEN 1 THEN
    MESSAGE "You are not the owner of this file or directory.".
  WHEN 2 THEN
    MESSAGE "The file or directory you want to delete does not exist.".
  OTHERWISE
    DISPLAY "OS Error #" + STRING(OS-ERROR,"99")
      FORMAT "x(13)" WITH FRAME b.
  END CASE.
```

Notes

- This function returns 0 if no error occurred.
- Use this function immediately following an OS-APPEND, OS-COPY, OS-CREATE-DIR, OS-DELETE, OS-RENAME, or SAVE CACHE statement to determine whether an error occurred during the statement's execution. If you do not, the next use of one of these statements overwrites the previous error code.
- Table 40 lists the Progress error codes that the OS-ERROR function can return.

Table 40: Progress OS-ERROR codes *(1 of 2)*

Error number	Description
0	No error
1	Not owner
2	No such file or directory
3	Interrupted system call
4	I/O error
5	Bad file number
6	No more processes
7	Not enough core memory
8	Permission denied
9	Bad address
10	File exists
11	No such device
12	Not a directory
13	Is a directory
14	File table overflow
15	Too many open files
16	File too large

Table 40: Progress OS-ERROR codes *(2 of 2)*

Error number	Description
17	No space left on device
18	Directory not empty
999	Unmapped error (Progress default)

See also OS-APPEND statement, OS-COPY statement, OS-CREATE-DIR statement, OS-DELETE statement, OS-RENAME statement, SAVE CACHE statement

OS-GETENV function

Returns a string that contains the value of the desired environment variable in the environment in which Progress is running.

Syntax

```
OS-GETENV ( environment-variable )
```

environment-variable

The name of the environment variable whose value you want to find.

Example

This procedure prompts a user for a report name. It then builds the full pathname where the report will be stored, using OS-GETENV to find the DLC directory. Finally, the procedure displays the full pathname.

r-os-env.p

```
DEFINE VARIABLE pathname AS CHARACTER
  FORMAT "x(32)"
  LABEL "The report will be stored in ".
DEFINE VARIABLE report_name AS CHARACTER
  FORMAT "x(32)"
  LABEL "Please enter report name." .

UPDATE report_name.
pathname = OS-GETENV("DLC") + "/" + report_name.
DISPLAY pathname WITH FRAME b SIDE-LABELS.
```

Notes

- If the environment variable is not defined, this statement returns the Unknown value (?).
- Since environment variables are case sensitive in some environments, make sure that the name you supply is the correct case.

OS-RENAME statement

Executes an operating system file rename or directory rename command from within Progress.

Syntax

```
OS-RENAME
  { source-filename | VALUE ( expression ) }
  { target-filename | VALUE ( expression ) }
```

source-filename

The name of the file or directory to rename.

VALUE (*expression*)

An expression that returns the name of the file or directory to rename. *expression* can contain constants, field names, and variable names.

target-filename

The new name of the file or directory.

VALUE (*expression*)

An expression that returns the new name of the file or directory. *expression* can contain constants, field names, and variable names.

Example

This procedure opens a dialog box that prompts the user to choose a file to rename. It then prompts for a new name. Finally, the procedure uses the OS-RENAME statement to rename the file.

r-os-nam.p

```
DEFINE VARIABLE sourcefile AS CHARACTER NO-UNDO.
DEFINE VARIABLE targetfile AS CHARACTER FORMAT "x(20)" VIEW-AS FILL-IN.
DEFINE VARIABLE OKpressed AS LOGICAL INITIAL TRUE.

Main:
REPEAT:

  SYSTEM-DIALOG GET-FILE sourcefile
    TITLE "Choose a File or Directory to Rename"
    MUST-EXIST
    USE-FILENAME
    UPDATE OKpressed.

  IF OKpressed = FALSE THEN
    LEAVE Main.
  UPDATE targetfile WITH FRAME newnameframe.
  OS-RENAME VALUE(sourcefile) VALUE(targetfile).
END.
```

Notes

- The filenames or directory names must conform to the naming conventions of the underlying operating system.
- If *source-filename* and *target-filename* specify different directories, this statement both renames the file and moves it to the new directory.
- Although an error can occur during execution of this statement, the statement does not generate an error message, raise an error condition, or affect the program's flow in any way. Check for an execution error by using the OS-ERROR function and evaluating the return.

See also

OS-ERROR function

OUTPUT CLOSE statement

Closes the default output destination or the output stream you name with the STREAM keyword in a prior OUTPUT TO statement.

Syntax

```
OUTPUT [ STREAM stream ] CLOSE
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

Example

This procedure sends customer data to a file by using the OUTPUT TO statement. All statements that normally send output to the terminal send output to the file named `cust.dat`. After all customer data is written to the file, the OUTPUT CLOSE statement resets the output destination, usually the terminal. The final DISPLAY statement displays Finished on the terminal.

r-out.p

```
OUTPUT TO cust.dat.

FOR EACH customer:
    DISPLAY cust-num name address address2 city state country SKIP(2)
    WITH 1 COLUMN SIDE-LABELS.
END.

OUTPUT CLOSE.
DISPLAY "Finished".
```

Notes

- The default output destination is the destination that was active when the procedure began. The output destination is usually the terminal unless the current procedure was called by another procedure while a different destination was active.
- A form feed (new page) is automatically output when a PAGED output stream is closed.
- If the output destination is the Windows clipboard, this statement writes all buffered output data to the clipboard in CF-TEXT format and clears the buffer.
- For more information on directing output, see *OpenEdge Development: Programming Interfaces*.

See also DEFINE STREAM statement, OUTPUT TO statement

OUTPUT THROUGH statement (NT, UNIX only)

Identifies a new output destination as the input to a process that Progress starts.

Syntax

```
OUTPUT [ STREAM stream ] THROUGH
  { program-name | VALUE ( expression ) }
  [ argument | VALUE ( expression ) ] ...
  [ ECHO | NO-ECHO ]
  [ MAP protermcap-entry | NO-MAP ]
  [ PAGED ]
  [ PAGE-SIZE { constant | VALUE ( expression ) } ]
  [ UNBUFFERED ]
  [     NO-CONVERT
    |  { CONVERT
           [ TARGET target-codepage ]
           [ SOURCE source-codepage ]
       }
  ]
```

STREAM *stream*

The name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

program-name

The name of the program to which you are supplying data from a procedure. This can be a standard command or your own program.

VALUE (*expression*)

An expression whose value is the name of a UNIX program to which you are supplying data from a procedure.

An expression is also the argument that you want to pass to the UNIX program. OUTPUT THROUGH passes the value of *expression* as a character string.

argument

An argument you want to pass to the UNIX program. The OUTPUT THROUGH statement passes this argument as a character string.

If the argument is the literal value paged, page-size, echo, no-echo, or unbuffered, you must enclose it in quotes to prevent Progress from using that argument as one of the PAGED, PAGE-SIZE, ECHO, NO-ECHO, or UNBUFFERED options for the OUTPUT THROUGH statement.

ECHO

Sends all input data read from a file to the UNIX program. Progress echoes data by default.

NO-ECHO

Suppresses the echoing of input data to the UNIX program.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* is an entry from the PROTERMCAP file. Use MAP to send output to a device that requires different character mappings than those in effect for the current output stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). Progress uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make Progress bypass character translation altogether. See *OpenEdge Deployment: Managing 4GL Applications* for more information on PROTERMCAP. See *OpenEdge Development: Internationalizing Applications* for more information on national language support.

PAGED

Formats the output into pages.

PAGE-SIZE { *constant* | VALUE (*expression*) }

Specifies the number of lines per page. The *expression* is a constant, field name, variable name, or expression whose value is an integer. The default number of lines per page is 56. If you use the TERMINAL option to direct output to the terminal, the default number of lines per page is the number of lines of TEXT widgets that fit on the screen. If you specify a non-zero value for PAGE-SIZE, then the PAGED option is assumed. If you specify PAGE-SIZE 0, the output is not paged.

UNBUFFERED

Writes one character at a time to a normally buffered data source, such as a file. Use the UNBUFFERED option only when you can intermingle your UNIX output (with the Progress UNIX statement) and your Progress output (with the OUTPUT THROUGH statement). That is, the OUTPUT THROUGH statement manages the buffering of output between the Progress procedure the UNIX program that it invokes, but it does not handle the buffering of output to any other programs that the Progress procedure might also invoke.

CONVERT

Allows you to modify the character conversions occurring between the UNIX program and Progress. By default, the OUTPUT TO statement converts characters from the code page specified with the Internal Code Page (-cpinternal) parameter to the code page specified with the Stream Code Page (-cpstream) parameter. If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name used in Progress memory (instead of -cpinternal). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the code page of the UNIX program (instead of -cpstream). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpinternal to -cpstream).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the external file and Progress. By default, the OUTPUT THROUGH statement converts characters from the -cpinternal code page to the -cpstream code page.

Examples

In this example, the customer names are displayed. This output is sent as input to the UNIX `wc` (word count) command. The output of `wc` is directed to the file wcdata using the standard UNIX redirection symbol (>). Finally, the results are displayed as three integers that represent the number of lines, words, and characters that were in the data sent to wc.

r-othru.p

```
OUTPUT THROUGH wc > wcdata.
/* word count UNIX utility */

FOR EACH customer:
    DISPLAY name WITH NO-LABELS NO-BOX.
END.

OUTPUT CLOSE.
PAUSE 1 NO-MESSAGE.
UNIX cat wcdata.
UNIX SILENT rm wcdata.
```

The `r-othru2.p` procedure uses the UNIX `crypt` program, which accepts lines of data, applies an algorithm based on an encryption key and writes the result to the UNIX standard output stream, that can be directed to a file. The output from the procedure is directed to `crypt`, which encrypts the customer names based on the password, mypass. The results of the encryption are stored in the ecust file. Then, Progress decrypts and displays this file.

r-othru2.p

```
OUTPUT THROUGH crypt mypass > ecust.

FOR EACH customer WHERE cust-num < 10:
  DISPLAY name WITH NO-LABELS NO-BOX.
END.

OUTPUT CLOSE.

UNIX crypt mypass <ecust.
```

Notes

- When you use the OUTPUT CLOSE statement to close an output destination used by an OUTPUT THROUGH statement, Progress closes the pipe, waits one second, and then continues.
- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that Progress uses for character management).
- If you specify a value of "undefined" for either *`source-codepage`* or *`target-codepage`*, no character conversion is performed.
- For more information on output destinations, see *OpenEdge Development: Programming Interfaces*.

See also DEFINE STREAM statement, OUTPUT CLOSE statement, OUTPUT TO statement

OUTPUT TO statement

Specifies an output destination.

Syntax

```
OUTPUT [ STREAM stream ] TO
  {     PRINTER [ printer-name ]
      | opsys-file
      | opsys-device
      | TERMINAL
      | VALUE ( expression )
      | "CLIPBOARD"
  }
  [ LOB-DIR { constant | VALUE ( expression ) } ]
  [ NUM-COPIES { constant | VALUE ( expression ) } ]
  [ COLLATE ]
  [ LANDSCAPE | PORTRAIT ]
  [ APPEND ]
  [ BINARY ]
  [ ECHO | NO-ECHO ]
  [ KEEP-MESSAGES ]
  [ NO-MAP | MAP protermcap-entry ]
  [ PAGED ]
  [ PAGE-SIZE { constant | VALUE ( expression ) } ]
  [ UNBUFFERED ]
  [     NO-CONVERT
    | { CONVERT
          [ TARGET target-codepage ]
          [ SOURCE source-codepage ]
      }
  ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

PRINTER **[** *printer-name* **]**

By default, this option sends output to the printer defined in the default print context. Specify a printer name to send output to a specific printer. Specifying a printer name overrides, but does not change, the printer defined in the default print context.

When you use this option, it implies that the device you are sending output to is paged, unless you also specify PAGE-SIZE 0.

In Windows, you must specify network printers in Universal Naming Convention format. For example:

```
\\fs_dev\hplas4
```

On UNIX, the printer spooling facilities (`lp` or `lpr`) are used automatically.

opsys-file

The name of a text file to which you want to direct output from a procedure. The file name can contain up to 255 characters.

opsys-device

Represents the name of an operating system device.

TERMINAL

Indicates that you want to direct output to the terminal. The terminal is the default output destination.

VALUE (*expression*)

Represents an expression whose value is the destination to which you want to send data.

"CLIPBOARD"(graphical interfaces only)

Specifies the system clipboard as the output destination. The quotes are required.

LOB-DIR { *constant* | VALUE (*expression*) }

Specifies the directory in which the EXPORT statement writes large object data files (such as BLOB and CLOB data files). The *constant* and *expression* arguments are character expressions that evaluate to an absolute pathname or a relative pathname (relative to the directory specified in *opsys-file*).

If the specified character expression evaluates to either the Unknown value (?) or a directory that does not exist, or you do not have permission to write to the specified directory, Progress raises the ERROR condition.

The LOB-DIR option is valid only when you specify an operating system file as the output destination.

NUM-COPIES { *constant* | VALUE (*expression*) }

Specifies the number of copies to print. The *constant* or *expression* parameters must evaluate to a positive integer. This option is supported in Windows only, and only with printer drivers that support multi-copy printing. Specifying the number of copies to print overrides, but does not change, the number of copies defined in the default print context.

The following statement prints three copies of each output page on the selected printer:

```
OUTPUT TO PRINTER NUM-COPIES 3.
```

COLLATE

Specifies whether multiple copies of output pages print in collated order. This option is supported in Windows only, and only with printer drivers that support collation.

LANDSCAPE

Specifies a landscape page orientation. This option is supported in Windows only, and only with printer drivers that support landscape page orientation. Specifying a page orientation overrides, but does not change, the page orientation defined in the default print context.

The following statement prints three copies of each output page with a landscape orientation on the selected printer:

```
OUTPUT TO PRINTER LANDSCAPE NUM-COPIES 3.
```

PORTRAIT

Specifies a portrait page orientation. This option is supported in Windows only, and only with printer drivers that support portrait page orientation. Specifying a page orientation overrides, but does not change, the page orientation defined in the default print context.

APPEND

Appends the output to the end of a file.

BINARY

Allows output to be written directly without any conversion or interpretation.

ECHO

Sends all input data read from a file to the output destination. Data is echoed by default.

NO-ECHO

Suppresses the echoing of input data to the output destination.

KEEP-MESSAGES

Causes the following messages not to echo to the default window: Progress error and warning messages, and messages from the MESSAGE statement. If you specify KEEP-MESSAGES, these messages are sent only to the output stream you specify.

MAP *protermcap-entry* | NO-MAP

The *protermcap-entry* value is an entry from the PROTERMCAP file. Use MAP to send output to a device that requires different character mappings than those in effect for the current output stream. Typically, *protermcap-entry* is a slash-separated combination of a standard device entry and one or more language-specific add-on entries (MAP laserwriter/french or MAP hp2/spanish/italian, for example). Progress uses the PROTERMCAP entries to build a translation table for the stream. Use NO-MAP to make Progress bypass character translation altogether. See *OpenEdge Deployment: Managing 4GL Applications* for more information on PROTERMCAP. See *OpenEdge Development: Internationalizing Applications* for more information on national language support.

PAGED

Formats the output into pages. Form feeds are represented by ^L (**CTRL+L**). When output is PAGED, a page break occurs every 56 lines. PAGED is automatic for output to a printer.

PAGE-SIZE { *constant* | VALUE (*expression*) }

Specifies the number of lines per page. The *expression* is a constant, field name, variable name, or expression whose value is an integer. The default number of lines per page is 56. If you are using the TERMINAL option to direct output to the terminal, the default number of lines per page is the number of lines of TEXT widgets that fit in the window. If you specify a non-zero value for *n*, then the PAGED option is assumed. If you specify PAGE-SIZE 0, the output is not paged in character mode; in a graphical interface, the default page size is used.

UNBUFFERED

Writes one character at a time to a normally buffered data source, such as a file. Use the UNBUFFERED option only when you can intermingle your UNIX output (with the Progress UNIX statement) and your Progress output (with the OUTPUT TO statement). That is, the OUTPUT TO statement manages the buffering of output between the Progress procedure the UNIX program that it invokes, but it does not handle the buffering of output to any other programs that the Progress procedure might also invoke.

CONVERT

Allows you to modify the character conversions occurring between the external file and memory. By default, the OUTPUT TO statement converts characters from the code page specified with the Internal Code Page (-cpinternal) parameter to the code page specified with the Stream Code Page (-cpstream) parameter . If you specify SOURCE *source-codepage* alone, the conversion accepts *source-codepage* as the code page name used in memory (instead of -cpinternal). If you specify TARGET *target-codepage*, the conversion accepts *target-codepage* as the code page of the external file (instead of -cpstream). If you specify both SOURCE *source-codepage* and TARGET *target-codepage*, it converts characters from the *source-codepage* to *target-codepage* (instead of -cpinternal to -cpstream).

TARGET *target-codepage*

Specifies the target code page of the character conversion (replacing -cpstream). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management).

SOURCE *target-codepage*

Specifies the source code page of the character conversion (replacing -cpinternal). The name that you specify must be a valid code page name available in the DLC/convmap.cp file (a binary file that contains all of the tables that Progress uses for character management).

NO-CONVERT

Specifies that no character conversions occur between the external file and memory. By default, the OUTPUT statement converts characters from the -cpinternal code page to the -cpstream code page.

Examples

The r-out.p procedure sends customer data to a file. The OUTPUT TO statement directs subsequent output to a file, so all statements that normally send output to the terminal send output to that file. After all the customer data has been displayed to the file, the OUTPUT CLOSE statement resets the output destination to its previous state, usually the terminal. The final DISPLAY statement displays Finished on the terminal because that is the new output destination.

r-out.p

```
OUTPUT TO cust.dat.

FOR EACH customer:
    DISPLAY cust-num name address address2 city state country SKIP(2)
        WITH 1 COLUMN SIDE-LABELS.
END.

OUTPUT CLOSE.
DISPLAY "Finished".
```

The r-termpg.p procedure sends customer data to the terminal. The OUTPUT TO TERMINAL PAGED statement directs output to the terminal in a paged format; all statements send output to the terminal one page at a time.

r-termpg.p

```
OUTPUT TO TERMINAL PAGED
DEFINE VAR x AS INTEGER.

FOR EACH customer BREAK BY sales-rep:
  FIND salesrep OF customer.
  FORM HEADER TODAY
       "Customer Listing For " to 43
       "Page " to 55 PAGE-NUMBER - x  TO 58 FORMAT "99"
       (salesrep.rep-name) FORMAT "x(30)" AT 25
       WITH FRAME hdr PAGE-TOP CENTERED.  VIEW FRAME hdr.
  DISPLAY cust-num COLUMN-LABEL "Customer!Number" name LABEL "Name"
          phone COLUMN-LABEL "Phone!Number" WITH CENTERED.
  IF LAST-OF (cust.sales-rep)
  THEN DO:
     x = PAGE-NUMBER.
     PAGE.
  END.
END.

OUTPUT CLOSE.
```

Notes

- The OUTPUT TO TERMINAL statement is the default unless the procedure was called by another procedure while a different output destination was active. The output destination at the beginning of the procedure is the current output destination of the calling procedure.
- The OUTPUT TO TERMINAL PAGED statement clears the screen and displays output on scrolling pages the length of the screen. Progress pauses before each page header. You can alter the pause using the PAUSE statement.
- Progress can display paged output to the terminal for frames that are wider than the width of the screen. The output is wrapped.
- To send output to a file correctly, you must specify the STREAM-IO option of the Frame phrase for any frame you use to write the file.
- If you send data to a file and you plan to use that data file later as input to a procedure, consider using the EXPORT statement. See the INPUT FROM statement reference entry for more information.

- If you send output to a device other than the terminal, ROW options in Frame phrases have no effect. ROW options also have no effect when you send output to a PAGED terminal. If you do not use the NO-BOX option with a Frame phrase, Progress omits the bottom line of the box, converts the top line to blanks, and ignores the sides of the box.
- All messages, including Compiler error messages and messages produced by the MESSAGE statement, are sent to the current output destination.
- If the field being output is MEMPTR, you must use the BINARY and NO-CONVERT mode of operation to prevent your data from becoming corrupted if it contains binary data.
- With the BINARY and NO-CONVERT options, you will not get a translation of new-lines to the appropriate characters for your operating system and there will be no code page conversion between `-cpinternal` and `-cpstream`.
- If the field being output is MEMPTR and your MEMPTR contains ASCII data you may want code page conversion. However, you cannot get conversion by using the CONVERT parameter on the MEMPTR. You can get code page conversion by using the MEMPTR with the GET-STRING and CODEPAGE-CONVERT functions and the PUT-STRING statement.
- On UNIX, if you want to use a print spooler with spooler options, you can use the Printer (-o) startup parameter to specify the options. See *OpenEdge Deployment: Startup Command and Parameter Reference* for more information on the Printer startup parameter.
- You must use a printer control sequence to change the number of lines per page produced by your printer.
- Unless otherwise specified, the OUTPUT TO PRINTER statement uses the default print context to determine the printer name, number of copies, and page orientation for a print job. If there is no default print context, Progress uses the printer control settings from the current environment.
- Use the SYSTEM-DIALOG PRINTER-SETUP statement to let users change the default print context through the Windows Print dialog box.
- Use the PRINTER-NAME attribute of the SESSION system handle to set the printer name in the default print context without user intervention.

- In Windows, the OUTPUT TO statement uses the PrinterFont settings in the current environment (either the Registry, or the [Startup] section of the initialization file) to define a font for a print job. The PrintFont settings are similar to the Font settings in the environment and take the following form:

```
PrinterFont [ n ] = facename
  [ , size = screen-point-size ]
```

  OUTPUT TO PRINTER uses the PrinterFont setting. OUTPUT TO LPT*n* uses the corresponding PrinterFont*n* entry. The *facename* parameter in a PrinterFont setting represents any valid Windows font supported on your system. If you specify a font that your printer does not support, printing might take a long time and yield unexpected results. The *screen-point-size* setting represents the point size, in screen units, for the font. Progress converts the point size to logical printer units.

- OUTPUT TO PRINTER in Windows performs the following processing:

  a) Checks the default print context. If there is no default print context, Progress checks the Windows printer control settings from the current environment. If no printer controls are set, Progress displays an error message and terminates the print operation.

  b) Checks the current environment (either the Registry, or the [Startup] section of the initialization file) for a PrinterFont setting. If there is a valid PrinterFont setting, Progress uses the font specified for the print job. If there is no PrinterFont setting or the setting specifies a non-existent font, Progress uses the default printer font for the job. If there is no point size specified for the font in the PrinterFont setting, Progress uses the default size for the printer.

- OUTPUT TO LPT*n* in Windows performs the following processing:

  a) Checks the ports settings in Windows for a definition of the specified LPT port. If there is no definition of the specified port, Progress displays an error message and terminates the print operation. If multiple definitions exist for a port, Progress uses the first definition that it finds.

b) Checks the current environment (either the Registry, or the [Startup] section in the initialization file) for a corresponding PrinterFont*n* setting (PrinterFont1 is for LPT1, etc.). If there is a valid corresponding PrinterFont*n* setting, Progress uses the font specified for the print job. If there is no corresponding PrinterFont*n* setting or the setting specifies a non-existent font, Progress uses the "courier new" font for the job and calculates the font height to fit 60 lines on a page. If there is no point size specified for the font in the PrinterFont*n* setting, Progress uses the default size for the printer.

c) Defines a header at the top of each page in the output. The size of the header is based upon the following calculation: 1.5 * *font-height*.

- In Windows only, OUTPUT TO "CLIPBOARD" buffers all output to the specified stream until the next OUTPUT CLOSE for that stream. The OUTPUT CLOSE statement then writes the output to the Windows clipboard in CF-TEXT format. You can buffer only up to 64K of data between any stream-related pair of OUTPUT TO "CLIPBOARD" and OUTPUT CLOSE statements. Any additional buffered data is lost.

  For information on providing additional clipboard reading and writing capabilities to your application, see *OpenEdge Development: Programming Interfaces* and the CLIPBOARD Handle reference entry in this manual.

- For any character conversions to occur, all of the necessary conversion tables must appear in `convmap.cp` (a binary file that contains all of the tables that Progress uses for character management).

- If you specify a value of “undefined” for either *source-codepage* or *target-codepage*, no character conversion is performed.

- The OpenEdge ADE toolset provides a portable solution for printing text files. The solution is a procedure called `_osprint.p` and it is located in the `adecomm` directory in the OpenEdge product directory (`DLC`). The `_osprint.p` procedure sends a specified text file to the default printer as paged output. For more information on the `_osprint.p` procedure, see *OpenEdge Development: Programming Interfaces*.

- For more information on changing your output destination, see *OpenEdge Development: Programming Interfaces*.

See also CLIPBOARD system handle, DEFINE STREAM statement, INPUT-OUTPUT CLOSE statement, PAGE-SIZE function, SESSION system handle, SYSTEM-DIALOG PRINTER-SETUP statement

OVERLAY statement

Overlays a character expression in a field or variable starting at a given position, and optionally for a given length.

Syntax

```
OVERLAY ( target , position [ , length [ , type ] ] )
  = expression
```

target

The name of the character field or variable that you want to overlay an *expression*.

position

An integer expression that indicates the first character position in *target* where you want to store *expression*. The value of *position* must be positive. If *position* is longer than *target*, Progress pads *target* with blanks to match *position*.

length

An integer expression that indicates the number of positions you want to allocate for the storage of *expression*. The *expression* is truncated or padded with blanks to match *length*. If you do not use the *length* argument or specify -1 as the length, OVERLAY uses the entire *expression*.

type

A character expression that directs Progress to interpret the specified *position* and *length* values as character units, bytes, or columns. A double-byte character registers as one character unit. By default, Progress interprets the specified *position* and *length* values as character units.

There are three valid types: "CHARACTER," "RAW," and "COLUMN." The expression "CHARACTER" specifies character units. The expression "RAW" specifies bytes. The expression "COLUMN" specifies display or print character-columns. If you specify the type as a constant expression, Progress validates the type specification at compile time. If you specify the type as a non-constant expression, Progress validates the type specification at run time.

expression

An expression that results in an integer value, constant, field name, variable name, or expression that results in a character string that you want to overlay on *target*. If you specify *length*, the *expression* is truncated or padded with blanks to match *length*.

Example

The `r-replc1.p` procedure lets you search for, and replace text strings in a paragraph in a window. When you run the procedure, you see the paragraph, which is an array with an extent of five. You also see a prompt. Enter the text string you want the system to search for, and the new text you want in its place. The procedure searches the paragraph, one line at a time, for the text you entered. The procedure uses the OVERLAY statement to replace the string of old text with the string of new text. The procedure also determines the length of the old text and the new text.

r-replc1.p

```
DEFINE VARIABLE chktext  AS CHARACTER.
DEFINE VARIABLE i        AS INTEGER.
DEFINE VARIABLE chkndx   AS INTEGER.
DEFINE VARIABLE ndx      AS INTEGER.
DEFINE VARIABLE old-text AS CHARACTER.
DEFINE VARIABLE new-text AS CHARACTER.
DEFINE VARIABLE max-len  AS INTEGER.
DEFINE VARIABLE comment  AS CHARACTER FORMAT "x(49)" EXTENT 5
  INITIAL ["You are probably interested in PROGRESS because",
    "you have a lot of information to organize.  You",
    "want to get at the information, add to it, and",
    "change it, without a lot of work and aggravation.",
    "You made the right choice with PROGRESS." ].

DISPLAY comment WITH CENTERED FRAME comm NO-LABELS
  TITLE "Why You Chose PROGRESS" ROW 4.

REPEAT:
    SET old-text LABEL "Enter text to search for"
    new-text LABEL "Enter text to replace with"
      WITH FRAME replace SIDE-LABELS CENTERED.
  max-len = MAXIMUM(LENGTH(old-text), LENGTH(new-text)).
  DO i = 1 TO 5:
    ndx = 1.
    DO ndx = 1 TO LENGTH(comment[i]):
      chktext = SUBSTRING(comment[i], ndx).
      chkndx = INDEX(chktext, old-text).
      IF chkndx <> 0 THEN DO:
        ndx = ndx + chkndx - 1.
        OVERLAY(comment[i], ndx, max-len, "CHARACTER") = new-text.
        ndx = max-len.
      END.
    END.
    DISPLAY comment[i] WITH FRAME comm.
  END.
END.
```

Notes

- The OVERLAY statement is not equivalent to the SUBSTRING statement. When you use the OVERLAY statement, a specified number of byte or character units within the target string are overlaid with the same number of units from an expression. Therefore, the length of the target string does not change. The SUBSTRING statement replaces a specified number of units within the target string with the entire value of an expression. This may change the length of the target string.
- Do not split double-byte characters. This statement allows you to overlay either the lead or trail-byte of the target string when you specify "RAW" as the *type* parameter.

See also SUBSTRING function, SUBSTRING statement

PAGE statement

Starts a new output page for PAGED output. No action is taken if output is already positioned at the beginning of a page.

Syntax

```
PAGE [ STREAM stream ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

Example

This procedure prints a customer report, categorized by state, and starts a new page for each state:

r-page.p

```
DEFINE VARIABLE laststate AS CHARACTER.

OUTPUT TO PRINTER.
FOR EACH customer BY state:
  IF state <> laststate THEN DO:
    IF laststate <> "" THEN PAGE.
    laststate = state.
  END.
  DISPLAY cust-num name address city state.
END.
```

Notes

- If the current output destination is not a paged device (you did not use the PAGED option in the OUTPUT TO statement), the PAGE statement has no effect.
- PAGE has no effect if you are already at the top of a new page.
- If any PAGE-TOP or PAGE-BOTTOM frames are active, they are output prior to the next display.
- See the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

See also

DEFINE STREAM statement, OUTPUT TO statement

PAGE-NUMBER function

Returns the page number of the output destination. If the output stream is not paged, PAGE-NUMBER returns a value of 0.

Syntax

```
PAGE-NUMBER [ ( stream ) ]
```

stream

The name of an output stream. If you do not name a stream, PAGE-NUMBER returns the page number of the default unnamed output stream.

Example

This procedure creates a customer report with a page number on each page:

r-pgnbr.p

```
OUTPUT TO pagenum.txt PAGED.
FOR EACH customer:
  FORM HEADER "Customer report" AT 30
    "Page:" AT 60 PAGE-NUMBER FORMAT ">>9" SKIP(1).
  DISPLAY cust-num name address city state country.
END.
```

See also

OUTPUT TO statement, PAGE statement

PAGE-SIZE function

Returns the page size (lines per page) of an output destination. If the output stream is not paged, PAGE-SIZE returns a value of 0.

Syntax

```
PAGE-SIZE [ ( stream ) ]
```

stream

The name of an output stream. If you do not name a stream, PAGE-SIZE returns the page size of the default unnamed output stream.

Example

This procedure prints a customer report categorized by state. At the end of each state category, it tests to see if there are at least four lines left on the page. The LINE-COUNTER function returns the current line number of output. If that number plus four is greater than the total number of lines on the page (returned by the PAGE-SIZE function), then the procedure skips to a new page. If there are four or more lines left, the procedure skips a line before printing the next customer record.

r-pgsize.p

```
OUTPUT TO PRINTER.

FOR EACH customer BREAK BY state:
  DISPLAY cust-num name address city state.
  IF LAST-OF(state) THEN DO:
    IF LINE-COUNTER + 4 > PAGE-SIZE
    THEN PAGE.
    ELSE DOWN 1.
  END.
END.
```

See also

OUTPUT THROUGH statement, OUTPUT TO statement

Parameter definition syntax

This syntax defines one or more run-time parameters of a user-defined function, or a method within a class (including a constructor method).

Note: To define run-time parameters in a Progress subprocedure, Windows dynamic link library (DLL) routine, UNIX shared library routine, or ActiveX control event procedure, see the DEFINE PARAMETER statement reference entry in this book.

```
( parameter [, parameter ] ... )
```

Use the following syntax to define each *parameter*, which can be scalar, array, class, interface, temporary table object, ProDataSet object, or buffer:

Syntax

```
{ INPUT | OUTPUT | INPUT-OUTPUT }
{   parameter-name AS data-type [ EXTENT [ expression ] ]
  | parameter-name AS [ CLASS ] { type-name }
  | TABLE temp-table-name [ APPEND ] [ BIND ]
  | TABLE-HANDLE temp-table-handle [ APPEND ] [ BIND ]
  | DATASET dataset-name [ APPEND ] [ BIND ]
  | DATASET-HANDLE dataset-handle [ APPEND ] [ BIND ]
  | BUFFER buffer-name FOR database-table-name
}
```

`INPUT | OUTPUT | INPUT-OUTPUT`

The parameter mode. An input parameter travels from the caller, which sets its value, to the function, which can use the value. An output parameter travels from the function, which sets its value, back to the caller, which can use the value. An input-output parameter travels from the caller, which sets its value, to the function, which can use and reset the value, then back to the caller, which can use the value.

parameter-name

The name of a scalar, array, class, or interface parameter.

AS *data-type*

The data type of a scalar or array parameter. Progress provides these data types: CHARACTER, COM-HANDLE, DATE, DATETIME, DATETIME-TZ, DECIMAL, HANDLE, INTEGER, LOGICAL, LONGCHAR, MEMPTR, RAW, RECID, ROWID, and WIDGET-HANDLE.

EXTENT **[** *expression* **]**

Specifies a determinate array parameter (which has a defined number of elements) or an indeterminate array parameter (which has an undefined number of elements). To define a determinate array parameter, specify the EXTENT option with the *expression* argument. This optional argument evaluates to an integer value that represents an extent for the array parameter. To define an indeterminate array parameter, specify the EXTENT option without the *expression* argument. Progress determines the size of indeterminate array parameters at runtime.

If you call a function that fixes the number of elements in an indeterminate array, Progress treats the fixed indeterminate array as a determinate array.

If you want to define a parameter that is like an array variable or field, using the LIKE option, but you do not want the parameter to be an array, you can use EXTENT 0 to indicate a non-array field.

If you are using the AS *datatype* option and you do not use the EXTENT option (or you specify *expression* as 0), the parameter is not an array parameter.

AS **[** CLASS **]** **{** *type-name* **}**

Specifies a class or interface parameter. Progress passes the object reference associated with the class or interface (by value), not the class or interface itself.

type-name

A character string that specifies the type name of a class or interface. Specify a type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

If the specified class or interface type name conflicts with an abbreviation of a built-in Progress data type name, such as INT for INTEGER, you must specify the CLASS keyword.

`TABLE` *temp-table-name*

Specifies a temporary table parameter.

You can pass a temporary table parameter to both local and remote procedures, as well as methods in a class. Progress passes the parameter by value, by default. That is, the calling routine and the called routine each have their own instance of the temporary table. When you invoke the function, Progress deep-copies the parameter from one routine's instance to the other routine's instance. Which table travels depends on whether the parameter is INPUT, OUTPUT, or INPUT-OUTPUT. When you pass a temporary table as an INPUT parameter, Progress overlays the stationary instance with the traveling table, by default. You can also append the traveling table to the end of the stationary instance by specifying the APPEND option. For more information about the APPEND option, see the option description below.

When passing a temporary table parameter to a user-defined function, you can override the default by passing the parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). Passing a temporary table parameter by reference or by binding allows the calling routine and the called routine to access the same object instance (instead of deep-copying the parameter).

For more information about passing a temporary table parameter by reference or by binding, see the Parameter passing syntax reference entry. For more information about temporary table parameters, see *OpenEdge Development: Progress 4GL Handbook*.

`TABLE-HANDLE` *temp-table-handle*

Specifies a temporary table handle parameter.

`DATASET` *dataset-name*

Specifies a ProDataSet object parameter.

You can pass a ProDataSet object parameter to both local and remote procedures, as well as methods in a class. Progress passes the parameter by value, by default. That is, the calling routine and the called routine each have their own instance of the object. When you invoke the function, Progress deep-copies the parameter from one routine's instance to the other routine's instance. Which parameter travels depends on whether the parameter is INPUT, OUTPUT, or INPUT-OUTPUT. When you pass a ProDataSet object as an INPUT parameter, Progress overlays the stationary instance with the traveling ProDataSet object, by default. You can also append the traveling ProDataSet object to the end of the stationary instance by specifying the APPEND option. For more information about the APPEND option, see the option description below.

When passing a ProDataSet object parameter to a user-defined function, you can override the default by passing the parameter by reference or by binding (that is, by specifying the parameter in a RUN statement using either the BY-REFERENCE or BIND option). Passing a ProDataSet object parameter by reference or by binding allows the calling routine and the called routine to access the same object instance (instead of deep-copying the parameter).

For more information about passing a ProDataSet object parameter by reference or by binding, see the Parameter passing syntax reference entry. For more information about ProDataSet object parameters, see *OpenEdge Development: Progress 4GL Handbook*.

`DATASET-HANDLE` *`dataset-handle`*

Specifies a ProDataSet object handle parameter.

`APPEND`

Specifies whether or not to append the traveling temporary table data to the stationary temporary table instance. To append input parameter data, specify the APPEND option in the FUNCTION statement. To append output parameter data, specify the APPEND option in the RUN statement.

`BIND`

Indicates that a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter binds a reference-only object in one routine to an object instance defined and instantiated in another local routine.

When you define a reference-only object in the calling routine, and you want to bind that object definition to an object instance in the called routine, define the parameter by specifying the BIND option in an INPUT or INPUT-OUTPUT parameter definition. When you define a reference-only object in the called routine, and you want to bind that object definition to an object instance in the calling routine, define the parameter by specifying the BIND option in an OUTPUT parameter definition. In either case, the reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates.

Caution: Do not delete the object or routine to which a reference-only object is bound, or you might be left with references to an object that no longer exists.

You can bind multiple reference-only object definitions to the same object instance. You can also bind a single reference-only object definition to the same object instance multiple times without generating an error. However, you cannot bind a single reference-only object definition to multiple object instances.

When passing one of these parameters to a remote procedure, Progress ignores the BIND option and deep-copies the parameter based on the specified parameter mode.

For more information about passing these parameters by binding, see the Parameter passing syntax reference entry.

`BUFFER` *`buffer-name`* `FOR` *`database-table-name`*

Specifies a database buffer parameter.

Note: A user-defined function that has one or more buffer parameters cannot be invoked remotely. For more information on remote user-defined functions, see *OpenEdge Application Server: Developing AppServer Applications*.

See also CONSTRUCTOR statement, FUNCTION statement, METHOD statement, SUPER function

Parameter passing syntax

This syntax specifies one or more parameters to pass to a Progress procedure, a user-defined function, or a method within a class (including a constructor method when instantiating a class object instance using the NEW statement).

```
( parameter [, parameter ] ... )
```

Use the following syntax to specify each *parameter*:

Syntax

```
INPUT | OUTPUT | INPUT-OUTPUT
 { expression
  | parameter-name AS data-type
  | parameter-name AS [ CLASS ] { type-name }
  | { { TABLE temp-table-name
      | TABLE-HANDLE temp-table-handle
      | DATASET dataset-name
      | DATASET-HANDLE dataset-handle
    } [ APPEND ] [ BY-VALUE | BY-REFERENCE | BIND ]
   }
 }
```

```
BUFFER buffer
```

`INPUT | OUTPUT | INPUT-OUTPUT`

The parameter mode. The default mode is INPUT.

expression

A constant, expression, field name, or variable name. A constant or expression can be only an INPUT parameter. A field name or variable name, which can receive a value, can be an OUTPUT or INPUT-OUTPUT parameter.

parameter-name

The name of the parameter.

AS *data-type*

The data type of the parameter.

AS **[** CLASS **]** **{** *type-name* **}**

Specifies a class or interface parameter. Progress passes the object reference associated with the class or interface (by value), not the class or interface itself.

type-name

A character string that specifies the type name of a class or interface. Specify a type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

If the specified class or interface type name conflicts with an abbreviation of a built-in Progress data type name, such as INT for INTEGER, you must specify the CLASS keyword.

TABLE *temp-table-name*

The name of a temporary table.

TABLE-HANDLE *temp-table-handle*

A handle to a temporary table.

Use a temporary table handle as a parameter for a dynamic temp-table object. The definition behind the handle and the contents of the temp-table are sent. The matching parameter definition for the temp-table handle may be either the dynamic TABLE-HANDLE option or the static TABLE option.

If you call a remote procedure asynchronously and pass a parameter as OUTPUT TABLE-HANDLE *temp-table-handle* APPEND, the event procedure must specify a corresponding DEFINE INPUT PARAMETER TABLE-HANDLE FOR *temp-table-handle* APPEND statement, and *temp-table-handle* must be global to both the calling procedure and the event procedure.

DATASET *dataset-name*

The name of a ProDataSet object.

DATASET-HANDLE *dataset-handle*

A handle to a ProDataSet object.

Use a ProDataSet object handle as a parameter for a dynamic ProDataSet object. The definition behind the handle and the contents of the ProDataSet object are sent. The matching parameter definition for the ProDataSet object handle may be either the dynamic DATASET-HANDLE option or the static DATASET option.

BUFFER *buffer*

The name of a buffer.

Note: You cannot pass BUFFER parameters to a remote procedure.

APPEND

Specifies whether or not to append the traveling temporary table data to the stationary temporary table data. To append output parameter data, specify the APPEND option in the RUN statement. To append input parameter data, specify the APPEND option in the DEFINE PARAMETER statement.

BY-VALUE | BY-REFERENCE | BIND

Specifies whether to pass a TABLE, TABLE-HANDLE, DATASET, or DATASET-HANDLE parameter by value, by reference, or by binding. The default is BY-VALUE.

You can pass TABLE, TABLE-HANDLE, DATASET, and DATASET-HANDLE parameters to both local and remote procedures. These parameter types are normally passed by value, by default. That is, the calling routine and the called routine each have their own instance of the object, and the parameter is deep-copied from the calling routine's instance to the called routine's instance.

When passing one of these parameters to a local routine, you can override the default in the calling routine by specifying the BY-REFERENCE or BIND option.

Passing one of these parameters to a local routine using the BY-REFERENCE option allows the calling routine and the called routine to access the same object instance. That is, both routines access the calling routine's instance and ignore the called routine's instance. Since the called routine's object instance is ignored, you should define the static object as reference-only by specifying the REFERENCE-ONLY option in the DEFINE statement for the object.

Passing one of these parameters to a local routine using the BIND option allows the calling routine and the called routine to access the same object instance. You can do this by:

- Binding a reference-only static object defined in one routine to an object instance defined in another routine.
- Binding an unknown TABLE-HANDLE or DATASET-HANDLE parameter defined in one routine to an object instance defined in another routine.

In the static case, you must define a reference-only object in either the calling routine or the called routine by specifying the REFERENCE-ONLY option in the DEFINE statement for the object. You must also define the parameter by specifying the BIND option in the DEFINE PARAMETER statement.

When you define a reference-only object in the calling routine and pass it to the called routine using the BIND option, Progress binds the definition of the object in the calling routine to the object instance in the called routine. When you define a reference-only object in the called routine and receive the object from the calling routine, Progress binds the definition of the object in the called routine to the object instance in the calling routine. In either case, the reference-only object definition remains bound to the object instance until the routine containing the reference-only object definition is deleted or terminates.

Caution: Do not delete the object or routine to which a reference-only object is bound, or you might be left with references to an object that no longer exists.

You can bind multiple reference-only object definitions to the same object instance. You can also bind a single reference-only object definition to the same object instance multiple times without generating an error. However, you cannot bind a single reference-only object definition to multiple object instances.

When passing one of these parameters to a remote procedure, Progress ignores the BY-REFERENCE and BIND options and deep-copies the parameter based on the specified parameter mode.

Note: You cannot pass DATASET or DATASET-HANDLE parameters to an asynchronous remote procedure.

For more information about passing parameters to both local and remote procedures, see *OpenEdge Development: Progress 4GL Handbook*.

See also FUNCTION statement, NEW statement, PUBLISH statement, RUN statement, RUN SUPER statement, SUPER( ) method, SUPER system reference

PAUSE statement

Suspends processing indefinitely, or for a specified number of seconds, or until the user presses any key.

Note: Does not apply to SpeedScript programming.

Syntax

```
PAUSE
  [ n ]
  [ BEFORE-HIDE ]
  [ MESSAGE message | NO-MESSAGE ]
  [ IN WINDOW window ]
```

n

A numeric expression specifying the number of seconds that you want to suspend processing. If you do not use this option, Progress suspends processing until the user presses any key.

BEFORE-HIDE

Specifies the pause action the user must take whenever frames are hidden automatically. If you specify *n*, *n* is the number of seconds Progress pauses before hiding. If you do not specify *n*, the pause lasts until the user presses a key.

MESSAGE *message*

Displays the message "Press spacebar to continue" on the status line of the terminal screen when Progress encounters a PAUSE statement. Use the MESSAGE option to override that default message. A *message* is a constant character string.

NO-MESSAGE

Tells Progress to pause but not to display a message on the status line of the terminal screen.

IN WINDOW *window*

Specifies the window to which the pause action applies. The value *window* must be a handle to a window. If you do not use the IN WINDOW phrase, the PAUSE statement applies to the current window.

Example

The FOR EACH block in this procedure reads each of the records from the customer table and displays information from each record. Because the DISPLAY uses a down frame (multiple records displayed in the frame), Progress usually fills the window with as many records as possible and then displays the message: “Press spacebar to continue”. The PAUSE 2 BEFORE-HIDE message tells Progress to pause only two seconds before hiding the frame and displaying additional records.

r-pause.p

```
PAUSE 2 BEFORE-HIDE MESSAGE
  "Pausing 2 seconds".
FOR EACH customer WITH 13 DOWN:
  DISPLAY cust-num name.
END.
```

Notes

- After you use PAUSE, that statement is in effect for all the procedures run in that session unless it is overridden by other PAUSE statements in those procedures, or until you return to the Editor.
- Using the PAUSE *n* BEFORE-HIDE statement is a good way to write a demonstration application that runs by itself.
- Progress automatically pauses before removing a frame and displays the “Press spacebar to continue” message if you have not had a chance to see the data in the frame.
- When a PAUSE occurs, Progress clears any keystrokes buffered from the keyboard, discarding any type-ahead characters.

PDBNAME function

Returns the physical name of a currently connected database.

Syntax

```
PDBNAME ( integer-expression | logical-name | alias )
```

integer-expression

If the parameter supplied to PDBNAME is an integer expression, and there are, for example, three currently connected databases, then PDBNAME(1), PDBNAME(2), and PDBNAME(3) return their physical names. Also, continuing the same example of three connected databases, PDBNAME(4), PDBNAME(5), etc., return the Unknown value (?).

logical-name | *alias*

This form of the PDBNAME function requires a quoted character string or a character expression as a parameter. If the parameter is the logical name of a connected database or an alias of a connected database, then the physical name is returned. Otherwise, it returns the Unknown value (?).

Example

This procedure finds the physical name of the database that currently has the DICTDB alias:

r-pdbnam.p

```
MESSAGE "The current DICTDB is" PDBNAME("DICTDB") + ".db".
```

Note

The old DBNAME function has been retained for compatibility and is equivalent to PDBNAME(1).

See also

ALIAS function, CONNECT statement, CONNECTED function, CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, SDBNAME function

PRESELECT phrase

Specifies a set of records to preselect for a DO or REPEAT block.

Syntax

```
PRESELECT
  [ EACH | FIRST | LAST ] record-phrase
  [ , [ EACH | FIRST | LAST ] record-phrase ] ...
  [ [ BREAK ]
      { BY expression [ DESCENDING ]
          | COLLATE ( string , strength [ , collation ] ) [ DESCENDING ]
      } ...
  ]
```

[EACH **|** FIRST **|** LAST **]** *record-phrase*

Goes through a table, selecting records that meet the criteria you specify in *record-phrase*. PRESELECT creates a temporary index that contains pointers to each of the preselected records in the database table. Then you can use other statements, such as FIND NEXT, within the block to process those records.

The *record-phrase* option identifies the criteria to use when preselecting records. Following is the syntax for the *record-phrase*:

```
{ record [ field-list ] }
[ constant ]
[ [ LEFT ] OUTER-JOIN ]
[ OF table ]
[ WHERE expression ]
[ USE-INDEX index ]
[ USING [ FRAME frame ] field
    [ AND [ FRAME frame ] field ] ...
]
[ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
[ NO-PREFETCH ]
```

Specifying multiple occurrences of *record-phrase* preselects the tables using an inner join. Also, any sorting you specify applies to all the tables. If you then do a FIND on the last table in the PRESELECT list, Progress reads records into the buffers for all of the tables in the list.

For more information on *record-phrase* and inner joins, see the Record phrase reference entry.

BREAK

When used in combination with the FIRST function, LAST function, FIRST-OF function, and LAST-OF function, BREAK indicates that subgroups are used for aggregation. If you use BREAK, you must also use BY.

BY *expression* **[** DESCENDING **]**

Sorts the preselected records by the value of *expression*. If you do not use the BY option, PRESELECT sorts the records in order by the index used to extract the records. The DESCENDING option sorts the records in descending order (not in the default ascending order).

COLLATE (*string* , *strength* **[** , *collation* **]**) **[** DESCENDING **]**

Generates the collation value of a string after applying a particular strength, and optionally, a particular collation. The DESCENDING option sorts the records in descending order (not in default ascending order).

string

A CHARACTER expression that evaluates to the string whose collation value you want to generate.

strength

A CHARACTER expression that evaluates to a Progress comparison strength or an International Components for Unicode (ICU) comparison strength.

The Progress comparison strengths include:

RAW — Generates a collation value for the string based on its binary value.

CASE-SENSITIVE — Generates a case-sensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, Progress applies the ICU TERTIARY strength.

CASE-INSENSITIVE — Generates a case-insensitive collation value for the string based on a particular collation. If you specify this strength with an ICU collation, Progress applies the ICU SECONDARY strength.

CAPS — Generates a collation value for the string based on its binary value after converting any lowercase letters in the string to uppercase letters, based on the settings of the Internal Code Page (`-cpinternal`) and Case Table (`-cpcase`) startup parameters.

The ICU comparison strengths include:

PRIMARY — Generates a collation value for the base characters in the string.

SECONDARY — Generates a collation value for the base characters and any diacritical marks in the string.

TERTIARY — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string.

QUATERNARY — Generates a case-sensitive collation value for the base characters and any diacritical marks in the string, and distinguishes words with and without punctuation. ICU uses this strength to distinguish between Hiragana and Katakana when applied with the ICU-JA (Japanese) collation. Otherwise, it is the same as TERTIARY.

IGNORE-SECONDARY — Generates a case-sensitive, but diacritically-insensitive, collation value for the string.

Note: Use ICU comparison strengths only with ICU collations.

collation

A CHARACTER expression that evaluates to the name of a Progress collation table or ICU collation. If *collation* does not appear, COLLATE uses the collation table of the client.

Progress reports an error and stops execution if one of the following occurs:

- *strength* does not evaluate to a valid value.
- *collation* does not evaluate to a collation table residing in the `convmap.cp` file.
- *collation* evaluates to a collation table that is not defined for the code page corresponding to the `-cpinternal` startup parameter.

Examples

To process a multi-table collection gathered by the PRESELECT option, use the last table named in the collection when you want to read the selected records. Progress then automatically retrieves records from the other tables.

r-presl1.p

```
REPEAT PRESELECT EACH order, customer OF order, EACH order-line OF order
            BY order.order-date BY order.cust-num BY order-line.item-num:
   FIND NEXT order-line.
   DISPLAY order.order-date order.cust-num customer.name
           order-line.item-num.
END.
```

The PRESELECT option in this example selects the logically joined record that consists of order, order-line, and customer, and makes all of these records available in the REPEAT block. Usually you perform more complex processing within the PRESELECT block.

If, within a PRESELECT block, you find a record using the ROWID of that record, Progress disregards any other selection criteria you applied to the PRESELECT. For example, suppose the ROWID of order number 4 is stored in the variable ord-rowid:

```
DO PRESELECT EACH order WHERE order-num > 5:
  FIND FIRST order WHERE ROWID(order) = ord-rowid.
  DISPLAY order.
END.
```

In this example, Progress finds and displays order number 4 even though the selection criteria specifies that the order number must be greater than 5. The ROWID always overrides other selection criteria. Furthermore, if you use FIND...WHERE ROWID(*record*) =..., the index cursor is not reset in the preselected list. That is, even if record ROWID(*record*) is in the preselected list, FIND NEXT does **not** find the record that follows it in the preselected list.

See also

DEFINE BUFFER statement, DO statement, FIND statement, REPEAT statement

PROC-HANDLE function

Returns a value in the appropriate data type (usually INTEGER) that is a unique identifier for a stored procedure.

Syntax

```
PROC-HANDLE
```

Example

This procedure runs the stored procedure pcust and writes the procedure handle to the variable handle. It writes the results of the stored procedure identified by this procedure handle into the Progress-supplied buffer, proc-text-buffer, and displays it.

```
DEFINE VAR handle AS INTEGER.

RUN STORED-PROCEDURE pcust handle = PROC-HANDLE
(10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer WHERE PROC-HANDLE = handle:
  DISPLAY proc-text.
END.
CLOSE STORED-PROCEDURE pcust WHERE PROC-HANDLE = handle.
```

Notes

- Progress Software recommends that you specify a procedure handle for each stored procedure that you run.
- You do not have to specify a handle if there is only one active stored procedure and you do not include SQL statements in the OpenEdge application. In the case of ORACLE only, the DataServer passes SQL statements to the ORACLE RDBMS and uses the default system handle in the process.

For more information on using this function, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

See also

CLOSE STORED-PROCEDURE statement, PROC-STATUS function, RUN STORED-PROCEDURE statement

PROC-STATUS function

Returns the return status from a stored procedure. The return status is an integer value that indicates whether a stored procedure failed and why.

Syntax

```
PROC-STATUS
```

Example

This procedure runs the ORACLE stored procedure pcust and writes the results of the stored procedure into the Progress-supplied buffer, proc-text-buffer. The CLOSE STORED-PROCEDURE statement then retrieves the output parameters. The return status is written to the variable stat and is displayed. This same code works for accessing a stored procedure from an ODBC-compliant data source:

```
DEFINE VAR stat AS INTEGER.

RUN STORED-PROCEDURE pcust (10, OUTPUT 0, OUTPUT 0).
FOR EACH proc-text-buffer:
END.
CLOSE STORED-PROCEDURE pcust stat = PROC-STATUS.
  DISPLAY stat.
```

Notes

- For descriptions of the possible values for the return status of a non-Progress stored procedure, see your non-Progress documentation.
- For more information on using this function, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

See also

CLOSE STORED-PROCEDURE statement, PROC-HANDLE function, RUN STORED-PROCEDURE statement

PROCEDURE statement

Declares an internal procedure.

Syntax

```
PROCEDURE proc-name
  [ EXTERNAL "dllname"
    [ CDECL | PASCAL | STDCALL ]
    [ ORDINAL n ]
    [ PERSISTENT ]
  ]
  [ PRIVATE ]
  [ IN SUPER ]
```

proc-name

The name of the internal procedure.

EXTERNAL "*dllname*"

Identifies the internal procedure as a Windows dynamic link library (DLL) routine or a UNIX shared library routine. The *dllname* argument, specified as a string literal, is the name of the DLL or library containing the routine.

CDECL

Tells Progress to use the C calling convention when accessing the routine.

PASCAL

Supported only for backward compatibility.

STDCALL

Tells Progress to use the standard Windows calling convention when accessing the routine. This is the default.

ORDINAL *n*

Specifies the number of the DLL entry point (the *n*th routine) to invoke. If you use the ORDINAL option, then *proc-name* can specify any name used in the corresponding RUN statement to reference the routine. If you omit the ORDINAL option, *proc-name* specifies which DLL routine you want to invoke.

PERSISTENT

Specifies that the DLL or shared library routine should remain loaded in memory until Progress exits or the session executes the RELEASE EXTERNAL statement.

PRIVATE

Indicates the following about the internal procedure:

- That it cannot be invoked from an external procedure—that is, from a procedure file external to the current procedure file.
- That the INTERNAL-ENTRIES attribute on the procedure that defines it does not provide its name (unless the procedure that defines it is the current procedure file).
- That the GET-SIGNATURE method on the procedure that defines it does not provide its signature (unless the procedure that defines it is the current procedure file).

IN SUPER

Indicates that the definition of the internal procedure resides in a super procedure. For more information on super procedures, see *OpenEdge Development: Progress 4GL Handbook*.

Examples

The following example declares a Progress internal procedure that computes the factorial of an integer entered as an INPUT parameter. The result is returned as an OUTPUT parameter. Note that the following procedure calls itself recursively to obtain the result:

r-factrl.p

```
DEFINE VARIABLE FactorialResult AS INTEGER FORMAT ">>>,>>>,>>9".
DEFINE VARIABLE FactorialInput AS INTEGER.

REPEAT:
    SET FactorialInput
             VALIDATE(FactorialInput <= 12 AND FactorialInput >= 0,
                      "Value must be between 0 and 12.").
    RUN Factorial (INPUT FactorialInput, OUTPUT FactorialResult).
    DISPLAY FactorialResult.
END.

PROCEDURE Factorial:
   DEFINE INPUT PARAMETER  PTerm AS INTEGER.
   DEFINE OUTPUT PARAMETER FactorialResult AS INTEGER.

   DEFINE VARIABLE WorkingResult AS INTEGER.

   IF  PTerm <= 1 THEN DO:
      FactorialResult = 1.
      RETURN.
   END.
   ELSE DO:
      RUN Factorial (INPUT  PTerm - 1, OUTPUT WorkingResult).
      FactorialResult =  PTerm * WorkingResult.
   END.
END PROCEDURE.
```

The following example declares a DLL routine, `MessageBox()`, which displays a message:

r-dllex1.p

```
DEFINE VARIABLE result AS INTEGER.

MESSAGE "  It's a whole new world!"
  VIEW-AS
    ALERT-BOX MESSAGE
    BUTTONS OK
    TITLE "Progress Message".

RUN MessageBoxA (0, "  It's a whole new world, again!!",
  "Progress DLL Access", 0, OUTPUT result).

PROCEDURE MessageBoxA EXTERNAL "user32.dll":
  DEFINE INPUT PARAMETER hwnd AS LONG.
  DEFINE INPUT PARAMETER mbtext AS CHARACTER.
  DEFINE INPUT PARAMETER mbtitle AS CHARACTER.
  DEFINE INPUT PARAMETER style AS LONG.
  DEFINE RETURN PARAMETER result AS LONG.
END.
```

Notes

- You can terminate a PROCEDURE statement with either a period (.) or a colon(:).
- You can place an internal procedure before, after, or in the middle of your main procedure code. You **cannot** nest an internal procedure within another internal procedure.
- Use the RUN statement to invoke an internal procedure. You can run an internal procedure from within the external procedure that defines it, either from the main-line of the external procedure or from another internal procedure defined in the external procedure. You can also run an internal procedure defined in another external procedure using the IN *proc-handle* option of the RUN statement as long as the external procedure meets one of these conditions:
  - It is active on the procedure call stack.
  - It is an instance of a persistent procedure.

- If you declare the internal procedure as a Progress 4GL procedure, the body of the procedure can contain zero or more 4GL statements. These statements can include definitions of run-time parameters (using the DEFINE PARAMETER statement), local program variables, frames, widgets, and buffers. Any such objects you define within the internal procedure remain in effect only for the life of the internal procedure.

  If you are defining the internal procedure for use as an event procedure to handle asynchronous remote requests, you can specify run-time parameters as INPUT only. (Any other type of parameter generates a run-time error.) Each INPUT parameter must correspond in order and data type with an OUTPUT (or INPUT-OUTPUT) parameter as defined in the remote procedure that executes the request. For more information on working with asynchronous remote requests and event procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

- You cannot define shared objects, work tables, or temporary tables within an internal procedure.

- An internal procedure can reference any objects defined in the outer procedure block. For example, it can reference variables, buffers (explicit or implicit; shared or unshared), variables, run-time parameters, named frames, or temporary tables. If you define an object with the same name in the internal procedure and the external procedure, a reference within the internal procedure resolves to the local object.

- A buffer explicitly defined in an internal procedure is scoped to the internal procedure. Any other buffers are scoped to the outer procedure block.

- If you declare the internal procedure as a DLL or UNIX shared library routine (use the EXTERNAL option), the body of the procedure can contain zero or more DEFINE PARAMETER statements.

- For more information on accessing DLL or UNIX shared library routines from Progress, see the chapter on DLLs in *OpenEdge Development: Programming Interfaces*.

- To define the internal procedure as an event handler for ActiveX controls (OCX event procedure), you must specify *proc-name* according to the following syntax:

```
{    control-frame-name .control-name .event-name
   |  ANYWHERE .event-name
}
```

  In *control-frame-name.control-name.event-name*, *control-frame-name* is the name (unquoted) of the control-frame that contains the ActiveX control. This is the name that the AppBuilder typically assigns to the control-frame (NAME widget attribute) when you insert the control into your user interface. The *control-name* is the value (unquoted) that you assign to the control Name property at design time in the AppBuilder Property Window. The *event-name* is the name (unquoted) of the ActiveX control event that you want to trigger execution of this procedure.

  In ANYWHERE.*event-name*, ANYWHERE specifies an event procedure that handles the specified event in any ActiveX control. This event procedure executes only if you have not defined a *control-frame-name.control-name.event-name* event procedure that exactly matches the control/event combination at run time.

  At design time, the AppBuilder lists the available events for a control and automatically creates a template for the OCX event procedure definition from the event that you select. For more information on how to create OCX event procedures in the AppBuilder, see the information on ActiveX controls in *OpenEdge Development: Programming Interfaces*. For more information on how to work with OCX event procedures in an application, see *OpenEdge Development: Programming Interfaces*.

- When you define an OCX event procedure, you can access the component handle (COM-HANDLE value) of the control that generates the event at run time using the COM-SELF system handle. You can also access the widget handle of the parent control-frame using the SELF system handle.

- The RETURN-VALUE function provides the value returned by the most recently executed RETURN statement of a local or remote procedure.

- For SpeedScript, the only invalid option is PASCAL.
- You can declare an internal procedure as a routine in a UNIX shared library in the same manner as declaring a DLL routine. The one exception is that the ORDINAL option is not applicable to UNIX and will be ignored. For example:

```
PROCEDURE atoi EXTERNAL "/usr/lib/libc.so.1":
. . .
```

- On 64-bit platforms, long integers passed from a shared library to Progress will lose their upper four bytes.

See also COM-SELF system handle, DEFINE PARAMETER statement, END statement, RUN statement, TRIGGER PROCEDURE statement

PROCESS EVENTS statement

Processes all outstanding events without blocking for user input.

Syntax

```
PROCESS EVENTS
```

Example

This procedure counts to 1,000 until you choose **STOP**:

r-proevs.p

```
DEFINE BUTTON stop-it LABEL "STOP".
DEFINE VARIABLE i AS INTEGER.
DEFINE VARIABLE stop-sel AS LOGICAL INITIAL FALSE.
DISPLAY stop-it.

ON CHOOSE OF stop-it
   stop-sel = TRUE.

ENABLE stop-it.

DO i = 1 TO 1000:
  DISPLAY i VIEW-AS TEXT.
  PROCESS EVENTS.
  IF stop-sel
  THEN LEAVE.
END.
```

On each pass through the loop, the procedure displays the new value of i and then checks whether any events are waiting to be processed. If no events have occurred, execution continues and the loop iterates. If the STOP button has been chosen, that event is processed changing the value of stop-sel. When execution continues, the program exits the loop.

If the loop does not contain the PROCESS EVENTS statement, the choose event never processes and the loop iterates until i equals 1,000.

Notes

- The WAIT-FOR statement processes all pending events and blocks all other execution until a specified event occurs. The PROCESS EVENTS statement processes all pending events and immediately continues execution with the next statement.
- If there are any asynchronous requests for which PROCEDURE-COMPLETE events have been received but not yet processed, this statement processes these events as described for the WAIT-FOR statement.

See also

WAIT-FOR statement

PROGRAM-NAME function

Returns the name of the calling program.

Syntax

```
PROGRAM-NAME( n )
```

n

The numeric argument. If *n* is 1, the name of the current program is returned. If *n* is 2, the name of the calling program is returned. If there is no calling program then you have reached the top of the call stack and Progress returns the Unknown value (?).

Example

This procedure returns the names of any procedure(s) that called it, and displays the number of levels that the procedure was nested:

r-prgnm.p

```
/* Note this program should be run as a subroutine */
/* The deeper the nesting, the better the illustration */

DEFINE VAR level AS INT INITIAL 1.
REPEAT WHILE PROGRAM-NAME(level) <> ?.
DISPLAY LEVEL PROGRAM-NAME(level) FORMAT "x(30)".
level = level + 1.
END.
```

Notes

- If you execute a procedure directly from the Procedure Editor or the User Interface Builder, then PROGRAM-NAME(1) returns the name of a temporary file rather than the name of the actual procedure file.

- The PROGRAM-NAME function is useful when developing on-line help. For example, you can use the following code in your help routine to produce a program trace:

r-trace.p

```
DEFINE VARIABLE i     AS INTEGER.
DEFINE VARIABLE plist AS CHARACTER FORMAT "x(70)".

FORM
  plist
  WITH FRAME what-prog OVERLAY ROW 10 CENTERED 5 DOWN NO-LABELS
     TITLE " Program Trace ".

i = 2. /* Skip the current routine: PROGRAM-NAME(1) */
DO WHILE PROGRAM-NAME(i) <> ?:
   IF i = 2
   THEN plist = "Currently in       : " + PROGRAM-NAME(i).
   ELSE plist = "Which was called by: " + PROGRAM-NAME(i).

   i = i + 1.
   DISPLAY plist WITH FRAME what-prog.
   DOWN WITH FRAME what-prog.
END.

PAUSE.
HIDE FRAME what-prog.
```

- If the procedure you reference is an internal procedure, then PROGRAM-NAME returns a string with the following form:

```
"internal-procedure-name source-file-name"
```

- If the procedure you reference is a user interface trigger associated with a widget, then PROGRAM-NAME returns a string with the following form:

```
"USER-INTERFACE-TRIGGER source-file-name"
```

- If the procedure you reference is a user interface trigger that uses the ANYWHERE keyword, then PROGRAM-NAME returns a string with the following form:

```
"SYSTEM-TRIGGER source-file-name"
```

- If the procedure you reference is a session database trigger, then PROGRAM-NAME returns a string with the following form:

```
"type-TRIGGER source-file-name"
```

  Where *type* is either ASSIGN, CREATE, DELETE, FIND, or WRITE.

- If the call stack contains a method reference, then PROGRAM-NAME returns a string with the following form:

```
"method-name class-file-name"
```

  Where *class-file-name* is the name of the class definition (`.cls`) file in which *method-name* is implemented.

PROGRESS function

Returns one of the following character values which identifies the Progress product that is running: Full, Query or Run-Time. Can also return COMPILE if you use the Developer's Toolkit, or COMPILE-ENCRYPT if you use the run-time Compiler.

Note: Does not apply to SpeedScript programming.

Syntax

```
PROGRESS
```

Examples

The following procedure uses the PROGRESS phrase function to determine which exit prompt is displayed on a menu.

r-progfn.p

```
/* Depending on the version of PROGRESS you are running, */
/* the main menu reflects available features for end-user */

DEFINE VARIABLE menu AS CHARACTER EXTENT 3.
DEFINE VARIABLE exit-prompt AS CHARACTER.

IF PROGRESS EQ "FULL" THEN
  exit-prompt = " 3. Return to Full Editor ".
ELSE IF PROGRESS EQ "QUERY" THEN
  exit-prompt = " 3. Return to Query Editor".
ELSE IF PROGRESS EQ "RUN-TIME" THEN
  exit-prompt = "3. Exit Program".

DO WHILE TRUE:
  DISPLAY
    "  1. Display Customer Data" @ menu[1] SKIP
    "  2. Display Order Data"    @ menu[2] SKIP
    exit-prompt                  @ menu[3]
    FORMAT "x(26)" SKIP
  WITH FRAME choices NO-LABELS.

  CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
    TITLE "Demonstration menu" CENTERED ROW 10.
  HIDE FRAME choices.
  IF FRAME-INDEX EQ 1 THEN MESSAGE
    "You picked option 1.".
  ELSE IF FRAME-INDEX EQ 2 THEN MESSAGE
    "You picked option 2.".
  ELSE IF FRAME-INDEX EQ 3 THEN RETURN.
END.
```

This procedure displays a message that tells you the type of Progress product you are using:

r-prodct.p

```
MESSAGE "You are currently running this PROGRESS product:" PROGRESS
  VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
```

See also

DBVERSION function, PROVERSION function

PROMPT-FOR statement

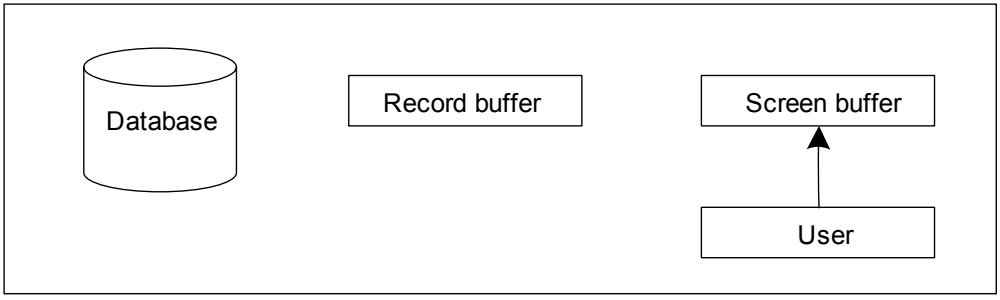
Requests input and places that input in the screen buffer (frame).

The PROMPT-FOR statement is a combination of the following statements:

- ENABLE Enables the specified field-level widgets (in this case fill-in fields) for input.
- WAIT-FOR Blocks for input and processes all Progress events until a specific Progress event occurs, in this case the GO universal key function event.
- DISABLE Disables the specified field-level widgets (in this case fill-in fields) for input.

Note: Does not apply to SpeedScript programming.

Data movement



Syntax

```
PROMPT-FOR
  [ STREAM stream ]
  [ UNLESS-HIDDEN ]
  {     { field
          [ format-phrase ]
          [ WHEN expression ]
        }
    |   { TEXT ( { field
                   [ format-phrase ]
                   [ WHEN expression ]
                 } ...
               )
        }
    |   { constant
          [ { AT | TO } n ]
          [ VIEW-AS TEXT ]
          [ FGCOLOR expression ]
          [ BGCOLOR expression ]
          [ FONT expression ]
        }
    |   SPACE [ ( n ) ]  |  SKIP [ ( n ) ]  |  ^
  } ...
  [ GO-ON ( key-label ... ) ]
  [ IN WINDOW window ]
  [ frame-phrase ]
  [ editing-phrase ]
```

```
PROMPT-FOR
  [ STREAM stream ]
  [ UNLESS-HIDDEN ]
  record [ EXCEPT field ... ]
  [ IN WINDOW window ]
  {  [ frame-phrase ] }
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

UNLESS-HIDDEN

Restricts PROMPT-FOR to fields whose HIDDEN attribute is FALSE.

field

Specifies the name of the field or variable whose value you want to enter and store in the screen buffer. Remember that the PROMPT-FOR statement only accepts input and stores it in the screen buffer. The underlying record buffer of a field or variable is unaffected.

This *field* parameter is demonstrated in the following program:

```
DEFINE VARIABLE x AS INTEGER INITIAL 3.
PROMPT-FOR x.
MESSAGE "Record buffer" x SKIP(0) "Screen buffer" INPUT x.
```

The program does the following:

- Stores the initial value of x in a record buffer.
- Prompts for a new value of x, and stores the new value in a screen buffer.
- Displays the value in the record buffer, retrieves the value in the screen buffer, then displays that.

In the case of array fields, array elements with constant subscripts are treated just like any other field. Array fields with no subscripts or in the FORM statement are expanded as though you had typed in the implicit elements. See the DISPLAY statement reference entry for information on how array fields with expressions as subscripts are handled.

format-phrase

Specifies one or more frame attributes for a field, variable, or expression. For more information on *format-phrase*, see the Format phrase reference entry.

WHEN *expression*

Prompts for the field only when *expression* has a value of TRUE. Here, *expression* is a field name, variable name, or expression that evaluates to a LOGICAL value.

TEXT

Defines a group of character fields or variables (including array elements) to use automatic word-wrap. The TEXT option works only with character fields. When you insert data in the middle of a TEXT field, Progress wraps data that follows into the next TEXT field, if necessary. If you delete data from the middle of a TEXT field, Progress wraps data that follows into the empty area.

If you enter more characters than the format for the field allows, Progress discards the extra characters. The character fields must have formats of the form x(n). A blank in the first column of a line marks the beginning of a paragraph. Lines within a paragraph are treated as a group and will not wrap into other paragraphs.

Table 41 lists the keys you can use within a TEXT field and their actions.

Table 41: Key actions in a TEXT field *(1 of 2)*

Key	Action
APPEND-LINE	Combines the line the cursor is on with the next line.
BACK-TAB	Moves the cursor to the previous TEXT field.
BREAK-LINE	Breaks the current line into two lines beginning with the character the cursor is on.
BACKSPACE	Moves the cursor one position to the left and deletes the character at that position. If the cursor is at the beginning of a line, **BACKSPACE** moves the cursor to the end of the previous line.
CLEAR	Clears the current field and all fields in the TEXT group that follow.
DELETE-LINE	Deletes the line the cursor is on.
NEW-LINE	Inserts a blank line below the line the cursor is on.
RECALL	Clears fields in the TEXT group and returns initial data values for the group.

Table 41: Key actions in a TEXT field *(2 of 2)*

Key	Action
RETURN	If you are in overstrike mode, moves to the next field in the TEXT group on the screen. If you are in insert mode, the line breaks at the cursor and the cursor is positioned at the beginning of the new line.
TAB	Moves to the field after the TEXT group on the screen.If there is no other field, the cursor moves to the beginning of the TEXT group.

In this procedure, the s-com, or Order Comments field is a TEXT field. Run the following procedure and enter text in the field to see how the TEXT option works:

r-text.p

```
DEFINE VARIABLE s-com AS CHARACTER FORMAT "x(40)" EXTENT 5.

FORM "Shipped   :" order.ship-date AT 13 SKIP
     "Misc Info :" order.instructions AT 13 SKIP(1)
     "Order Comments :" s-com AT 1
WITH FRAME o-com CENTERED NO-LABELS TITLE "Shipping Information".

FOR EACH customer, EACH order OF customer:
    DISPLAY cust.cust-num cust.name order.order-num order.order-date
            order.promise-date WITH FRAME order-hdr CENTERED.
    UPDATE ship-date instructions TEXT(s-com) WITH FRAME o-com.
    s-com = "".
END.
```

constant [AT n | TO *n*] [VIEW-AS TEXT] [FGCOLOR *expression*] [BGCOLOR *expression*] [FONT *expression*]

Specifies a literal value that you want displayed in the frame. If you use the AT option, *n* is the column in which you want to start the display. If you use the TO option, *n* is the column in which you want to end the display. You can use the BGCOLOR, FGCOLOR, and FONT options to define the colors and font in which the constant is displayed. If you use the VIEW-AS TEXT option, the constant is displayed as a text widget rather than a fill-in field.

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces to insert after the field is displayed. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, a new line is started and any extra spaces are discarded. If you do not use this option or *n*, one space is inserted between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines to insert after the field is displayed. The *n* can be 0. If you do not use this option, Progress does not skip a line between expressions unless the expressions do not fit on one line. If you use the SKIP option, but do not specify *n*, or if *n* is 0, Progress starts a new line unless it is already at the beginning of a new line.

^

Tells Progress to ignore an input field when input is being read from a file. Also, the following statement will read a line from an input file and ignore that line. This is an efficient way to skip over lines.

```
PROMPT-FOR ^
```

GO-ON (*key-label* . . .)

The GO-ON option tells Progress to execute the GO action when the user presses any of the keys listed. The keys you list are used in addition to keys that perform the GO action by default (such as **F1** or **RETURN** on the last field) or because of ON statements.

When you list a key in the GO-ON option, you use the keyboard label of that key. For example, if you want Progress to take the GO action when the user presses **F2**, you use the statement GO-ON(**F2**). If you list more than one key, separate them with spaces, not commas.

IN WINDOW *window*

Specifies the window in which the prompt occurs. The expression *window* must resolve to a handle to a window.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

editing-phrase

Supported only for backward compatibility.

Identifies processing to take place as each keystroke is entered. This is the syntax for *editing-phrase*:

```
[ label : ] EDITING: statement ... END
```

For more information on *editing-phrase*, see the EDITING phrase reference entry.

record

The name of a record buffer. All of the fields in the record will be processed exactly as if you prompted for each of them individually.

To use PROMPT-FOR with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

EXCEPT *field*

Affects all fields except those listed in the EXCEPT phrase.

Examples

The `r-prmpt.p` procedure requests a customer number from the user and stores that number in the screen buffer. The FIND statement reads a record from the customer database table.

r-prmpt.p

```
REPEAT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING cust-num NO-ERROR.
    IF NOT AVAILABLE customer THEN DO:
      MESSAGE "No such customer number." .
      UNDO, RETRY.
    END.
    DISPLAY name phone sales-rep.
END.
```

The r-prmpt2.p procedure requests the initials of a sales representative and stores those initials in the screen buffer. The FIND statement uses the initials stored in the screen buffer to read a record from the salesrep database table. After finding the record, the procedure displays sales rep information.

r-prmpt2.p

```
REPEAT:
  PROMPT-FOR salesrep.sales-rep LABEL "Sales rep's initials"
    WITH FRAME namefr ROW 2 SIDE-LABELS.
  FIND salesrep USING sales-rep.
  DISPLAY rep-name region month-quota WITH 1 DOWN NO-HIDE.
END.
```

Notes

- PROMPT-FOR puts user-supplied data into a screen buffer. It does not put any data into a record buffer. Therefore, if you want to use the data in the screen buffer, you must use the INPUT function to refer to the data in the screen buffer or use the ASSIGN statement to move the data from the screen buffer into a record buffer. You can also use the USING option to FIND a record with the screen data index value.
- When Progress compiles a procedure, it designs all the frames used by that procedure. When it encounters a PROMPT-FOR statement, Progress designs the display of the prompt fields. When the procedure is run, the PROMPT-FOR statement puts data into those fields.
- If you are getting input from a device other than the terminal, and the number of characters read by the PROMPT-FOR statement for a particular field or variable exceeds the display format for that field or variable, Progress returns an error. However, if you are setting a logical field that has a format of "y/n" and the data file contains a value of YES or NO, Progress converts that value to "y" or "n".

See also DEFINE STREAM statement, EDITING phrase, Format phrase, Frame phrase

PROMSGS function

Returns the current value of the Progress PROMSGS variable.

Syntax

```
PROMSGS
```

Example

This example uses the PROMSGS function to determine whether the default message file (promsgs) is in use. If not, it uses the PROMSGS function again to display the name of the current message file.

r-promsg.p

```
IF PROMSGS = "promsgs"
THEN MESSAGE "Using default promsgs file.".
ELSE MESSAGE "Using" PROMSGS.
```

See also

PROMSGS statement

PROMSGS statement

Sets the Progress PROMSGS variable for the current OpenEdge session. The PROMSGS variable holds the name of the current Progress message file. Progress supplies different versions of this file to support various languages.

Syntax

```
PROMSGS = string-expression
```

string-expression

A character-string expression that resolves to the name of a Progress message file. You can specify a full or relative pathname for the messages file.

Example

This example prompts the user for a language name and then tries to find a message file for that language. If the message file is found, then the PROMSGS statement is used to make that the current message file. Subsequently, all Progress system messages are read from the new `promsgs` file. The PROMSGS function is used in an informative message.

r-swmsgs.p

```
DEFINE VARIABLE newlang AS CHARACTER FORMAT "x(16)"
  LABEL "Language" NO-UNDO.
DEFINE VARIABLE msgfile AS CHARACTER.

SET newlang HELP "Enter the new language for messages.".

IF newlang = "English"
THEN ASSIGN msgfile = "promsgs".
ELSE ASSIGN msgfile = "prolang/promsgs." +
  LC(SUBSTRING(newlang, 1, 3)).

IF SEARCH(msgfile) < > ?
THEN DO:
  PROMSGS = msgfile.
  MESSAGE "Messages will now be taken from" PROMSGS.
END.
ELSE DO:
  MESSAGE "Cannot find" msgfile.
    UNDO, RETRY.
END.
```

See also

PROMSGS function

PROPATH function

Returns the current value of the PROPATH environment variable.

Syntax

```
PROPATH
```

Example

This procedure first displays a comma-separated list of the directories in the current PROPATH. It then displays each directory in the current PROPATH, one per line.

r-ppath1.p

```
DEFINE VARIABLE i AS INTEGER.

DISPLAY PROPATH.

REPEAT i = 1 TO NUM-ENTRIES(PROPATH):
  DISPLAY ENTRY(i , PROPATH) FORMAT "x(30)".
END.
```

Notes

- Progress stores the PROPATH as a comma-separated list of directories. (Progress strips the operating-specific separation characters (a colon (:) on UNIX; a semicolon (;) in Windows) and replaces them with commas.
- The default format for PROPATH is x(70).
- For more information on the PROPATH environment variable, see its reference entry in *OpenEdge Getting Started: Installation and Configuration*.

See also

PROPATH statement

PROPATH statement

Sets the PROPATH environment variable for the current OpenEdge session.

When you start Progress, it automatically adds the $DLC directory and some subdirectories to your PROPATH. Progress always preserves these directories in your PROPATH, even if you change or clear your PROPATH. Thus, Progress can always find its executables and r-code.

Syntax

```
PROPATH = string-expression
```

string-expression

A field, variable, string constant, or combination of these that evaluates to a character string. The character string should be a list of directory paths. The directory names in the path can be separated by commas or by the appropriate separation character for your operating system. The directory pathnames can use the UNIX format for pathnames it a(/dir1/dir2/dir3 , for example) or the standard pathname format for your operating system. Use the slash-separated directory name format if you are concerned about portability across multiple operating systems.

Examples

The r-ppath.p procedure displays a strip menu with four choices. The procedure defines three arrays: menu holds the items for selection on the menu, proglist holds the names of the programs associated with the menu selections, and ppath holds the appropriate PROPATHs for each program. The CHOOSE statement allows the user to choose an item from the strip menu.

r-ppath.p

```
DEFINE VARIABLE menu AS CHARACTER EXTENT 4 FORMAT "X(20)"
  INITIAL ["1. Sales","2. Acctg","3. Personnel","4. Exit"].
DEFINE VARIABLE proglist AS CHARACTER EXTENT 4 FORMAT "X(8)"
  INITIAL ["sales.p","acctg.p","per.p","exit.p"].
DEFINE VARIABLE ppath    AS CHARACTER EXTENT 4 INITIAL
  ["sales/s-procs","acctg/a-procs","per/p-procs",","].

REPEAT:
  DISPLAY menu WITH TITLE " M A I N   M E N U " CENTERED
    1 COLUMN 1 DOWN NO-LABELS ROW 8 ATTR-SPACE.
  CHOOSE FIELD menu AUTO-RETURN.
  HIDE.
  PROPATH = ppath[FRAME-INDEX].
  RUN VALUE(proglist[FRAME-INDEX]).
END.
```

Progress uses the menu selection number as an index into the ppath and proglist arrays. Progress sets the PROPATH and runs the program.

This simple example changes and displays the PROPATH:

r-prpath.p

```
PROPATH=ENTRY(1,PROPATH) +
  ",/dlc,/dlc/proguide,/dlc/appl1/procs".
DISPLAY PROPATH.
```

Notes

- Changes to PROPATH last only for the current session. Any subprocesses inherit the PROPATH in effect when the OpenEdge session started.
- When you start Progress, it automatically adds the top directory of the Progress hierarchy and some subdirectories to your PROPATH. If you use the PROPATH statement to make a change, Progress adds the directories you specify to your existing PROPATH.
- Progress replaces separation characters in *expression* (a colon (:) on UNIX; a semicolon (;) in Windows) with commas, so the resulting PROPATH string can be accessed with the ENTRY function. Therefore, file pathnames passed in *expression* must not include embedded commas.
- If you change your PROPATH, and your old PROPATH included r-code libraries that are not in your new PROPATH, those libraries are automatically closed. If you run a procedure from a closed library, Progress displays an error message.
- For more information on the PROPATH environment variable, see *OpenEdge Getting Started: Installation and Configuration*.

See also ENTRY function, PROPATH function

PROVERSION function

Returns the version of Progress, or release of OpenEdge, you are running.

Note: The PROVERSION function returns the version or release number as a character string. If you do not convert the returned character values to integer values (and strip off any letters) before sorting multiple return values, the values will not sort as expected. For example, the OpenEdge 10.0 release sorts before the Progress 9.1 version.

Syntax

```
PROVERSION
```

Example

The following example displays your current Progress version or OpenEdge release:

r-vers.p

```
MESSAGE "You are currently running Version/Release" PROVERSION.
```

Notes

- The PROVERSION function is not supported in Progress versions earlier than 7. If you want to test whether a procedure is running under an earlier version, you can use the KEYWORD function to determine whether PROVERSION is a keyword in that version. For example:

  ```
  IF KEYWORD("PROVERSION") = ?
  THEN /* Lower than Version 7. */.
  ```

  After you have determined that PROVERSION is available in the current version, then you can call a subroutine to invoke PROVERSION.

- SpeedScript – Returns the WebSpeed version.

See also

DBVERSION function, PROGRESS function

PUBLISH statement

Causes a Progress named event to occur.

Note: Progress named events are completely different from the key function, mouse, widget, and direct manipulation events described in the "Events Reference" section on page 2171. For more information on Progress named events, see *OpenEdge Development: Progress 4GL Handbook*.

Syntax

```
PUBLISH event-name
  [ FROM publisher-handle ]
  [ ( parameter [ , parameter ] ... ) ]
```

event-name

A quoted character string or character expression representing the name of a named event. If you use a quoted character string, Progress adds *event-name* to the PUBLISHED-EVENTS attribute's list of events.

FROM *publisher-handle*

A procedure or widget handle representing the procedure or widget to which Progress attributes the named event.

The FROM option lets a procedure publish an event on behalf of another procedure or widget. For example, if you want procedure A to publish a named event on behalf of procedure B, set *publisher-handle* to the procedure handle of B.

If the FROM option does not appear, Progress attributes the event to THIS-PROCEDURE, the procedure that contains the PUBLISH statement.

Note: If the FROM option does not appear and the PUBLISH statement occurs in a nonpersistent procedure that does not publicize its handle, potential subscribers have no way of knowing the handle's value, and can subscribe to the event only by using the SUBSCRIBE statement's ANYWHERE option.

`( parameter [ , parameter ] ... )`

The parameters, if any, of the named event.

As in the RUN statement, you must supply a value for each INPUT and INPUT-OUTPUT parameter and a variable for each OUTPUT parameter.

Also, if a named event has one or more parameters, the PUBLISH statement and each subscriber's local internal procedure (which the SUBSCRIBE statement names and which Progress runs when the named event occurs) must specify identical signatures-where *signature* means the number of parameters and the data type and mode (INPUT, etc.) for each.

Note: When the named event occurs and Progress runs each subscriber's local internal procedure, if the signature of a local internal procedure does not match the signature in the PUBLISH statement, Progress reports a run time error. Since the PUBLISH statement runs with an implicit NO-ERROR, errors are stored in the ERROR-STATUS handle.

The parameter syntax is identical to that of the RUN statement. For its specification, see the Parameter passing syntax reference entry in this book.

Example

The following example consists of four procedure files: a driver, a publisher, and two subscribers. The driver, `r-nedrvr.p`, runs the publisher and the two subscribers persistently, then subscribes to the event NewCustomer on behalf of the second subscriber.

r-nedrivr.p

```
/* r-nedrvr.p  */
DEFINE VARIABLE hPub  AS HANDLE.
DEFINE VARIABLE hSub1 AS HANDLE.
DEFINE VARIABLE hSub2 AS HANDLE.

DEFINE BUTTON bNewCust LABEL "New Customer".
DEFINE BUTTON bQuit LABEL "Quit".

RUN r-nepub.p PERSISTENT set hPub.
RUN r-nesub1.p PERSISTENT set hSub1 (hPub).
RUN r-nesub2.p PERSISTENT set hSub2.

/* Subscribe to event NewCustomer on behalf of subscriber 2 */
SUBSCRIBE PROCEDURE hSub2 TO "NewCustomer" IN hPub.

FORM bNewCust bQuit WITH FRAME x.

ENABLE ALL WITH FRAME x.

ON CHOOSE OF bNewCust RUN NewCust in hPub.

WAIT-FOR CHOOSE OF bQuit OR WINDOW-CLOSE OF CURRENT-WINDOW.
```

The publisher, `r-nepub.p`, publishes the event NewCustomer:

r-nepub.p

```
/* r-nepub.p */
PROCEDURE NewCust:
  DEFINE VARIABLE name AS CHARACTER INITIAL "Sam".
  /* Let subscriber know new customer */
  PUBLISH "NewCustomer" (INPUT name).
END PROCEDURE.
```

The first subscriber, nesub1.p, subscribes to the event NewCustomer:

r-nesub1.p

```
/* r-nesub1.p */
DEFINE INPUT PARAMETER hPub AS HANDLE.
SUBSCRIBE TO "NewCustomer" IN hPub.

PROCEDURE NewCustomer:
  DEFINE INPUT PARAMETER name AS CHAR.
  MESSAGE "Subscriber 1 received event NewCustomer concerning" name
    VIEW-AS ALERT-BOX.
END.
```

The second subscriber, nesub2.p, already subscribed to the event NewCustomer, cancels all subscriptions:

r-nesub2.p

```
/* r-nesub2.p */
PROCEDURE NewCustomer:
  DEFINE INPUT PARAMETER name AS CHAR.
  MESSAGE "Subscriber 2 received event NewCustomer concerning" name
    VIEW-AS ALERT-BOX.
  /* This subscriber receives the first event, then removes itself */
  UNSUBSCRIBE TO ALL.
END.
```

To start the example, run the driver, r-nedrvr.p.

Notes

- If a named event has multiple subscribers, the order in which Progress notifies subscribers is undefined.
- INPUT-OUTPUT parameters can accumulate values from a set of subscribers. When a subscriber receives an INPUT-OUTPUT parameter, it has the value that the previous subscriber set it to. When the publisher receives an INPUT-OUTPUT parameter, it has the value that the last subscriber set it to.
- If a named event with multiple subscribers has OUTPUT parameters, each time a subscriber sets an OUTPUT parameter, Progress overwrites the previous value. For this reason, Progress Software Corporation recommends that you use OUTPUT parameters with named events only when there is a single subscriber.

- If a named event has multiple subscribers and several subscribers specify a RETURN statement with a return value, the RETURN-VALUE function evaluates to the return value set by the last subscriber.
- Progress executes the PUBLISH statement with an implicit NO-ERROR option. To find out if any errors occurred, and if so, which ones, use the ERROR-STATUS system handle.
- If *publisher-handle* is a widget handle, the value of SOURCE-PROCEDURE in each of the subscribers' internal procedures will be the handle of the procedure that created the widget.

See also PUBLISHED-EVENTS attribute, SUBSCRIBE statement, UNSUBSCRIBE statement

PUT CURSOR statement (Character only)

Makes the cursor visible on the screen at a specified position.

In data handling statements such as UPDATE, SET, PROMPT-FOR, and INSERT, the Progress 4GL handles cursor display so the user knows where the cursor is located in the window. However, if data is entered through the READKEY statement, and that statement is not part of an EDITING phrase, you might want to turn the cursor on so the user can see the location of the cursor while entering data.

Note: Does not apply to SpeedScript programming.

Syntax

```
PUT CURSOR
  {     OFF
      | { [ ROW expression ] [ COLUMN expression ] }
  }
```

OFF

Ends display of the cursor.

ROW *expression*

The row in which you want to display the cursor. In the ROW option, *expression* is a constant, field name, variable name, or expression whose value is an integer that indicates the row where you want to display the cursor. If you do not use the ROW option, PUT CURSOR does not reposition the cursor. Similarly, if you specify a ROW that is outside the screen area, Progress does not reposition the cursor.

COLUMN *expression*

The column in which you want to display the cursor. In the COLUMN option, *expression* is a constant, field name, variable name, or expression whose value is an integer that indicates the column where you want to display the cursor. If you do not use the COLUMN option, PUT CURSOR does not reposition the cursor. Similarly, if you specify a COLUMN that is outside the windows area, Progress does not repositions the cursor.

Example

The following procedure uses PUT CURSOR to make the cursor visible in an editor window. When you run the procedure, you see a frame in a window. You can type text into this frame. The procedure reads each key you enter and takes the appropriate action. Then PUT CURSOR places the cursor in the first row and the first column in the editing frame when you first run the procedure. As you type, the cursor continues to be visible. As the procedure passes through the REPEAT loop for each keystroke, it takes action based on each keystroke and moves the cursor as it takes the action.

The procedure stores the information you type in the comments array, one character at a time. When you finish typing, press **GO**. The procedure displays the array where Progress stored the typed information.

r-cursor.p *(1 of 3)*

```
DEFINE VARIABLE comment AS CHARACTER FORMAT "x(30)" EXTENT 4.
DEFINE VARIABLE r       AS INTEGER.
DEFINE VARIABLE c       AS INTEGER.
DEFINE VARIABLE lmargin AS INTEGER INITIAL 5.
DEFINE VARIABLE rmargin AS INTEGER INITIAL 34.
DEFINE VARIABLE ptop    AS INTEGER INITIAL 10.
DEFINE VARIABLE pbot    AS INTEGER INITIAL 13.
DEFINE VARIABLE r-ofst  AS INTEGER INITIAL 9.
DEFINE VARIABLE c-ofst  AS INTEGER INITIAL 4.
FORM SKIP(4) WITH WIDTH 32 ROW 9 COL 4 TITLE "Editor".

MESSAGE "Type text into the editor.  Press" KBLABEL("GO") "to end.".

VIEW.
r = ptop.
c = lmargin.
```

r-cursor.p *(2 of 3)*

```
REPEAT:
   PUT CURSOR ROW r COLUMN c.
   READKEY.
   IF KEYFUNCTION(LASTKEY) = "GO" THEN LEAVE.
   IF KEYFUNCTION(LASTKEY) = "END-ERROR" THEN RETURN.
   IF LASTKEY = KEYCODE("CURSOR-RIGHT") THEN DO:
      c = c + 1.
      IF c > rmargin
      THEN c = lmargin.
      NEXT.
   END.
   IF LASTKEY = KEYCODE("CURSOR-LEFT") THEN DO:
      c = c - 1.
      IF c < lmargin
      THEN c = rmargin.
      NEXT.
   END.
   IF LASTKEY = KEYCODE("CURSOR-DOWN") THEN DO:
      r = r + 1.
      IF r > pbot
      THEN r = ptop.
      NEXT.
   END.
   IF LASTKEY = KEYCODE("CURSOR-UP") THEN DO:
      r = r - 1.
      IF r < ptop
      THEN r = pbot.
      NEXT.
   END.
   IF LASTKEY = KEYCODE("RETURN") THEN DO :
       r = r + 1.
       IF r > pbot
       THEN r = ptop.
       c = lmargin.
       NEXT.
   END.
```

r-cursor.p *(3 of 3)*

```
   IF LASTKEY = KEYCODE("BACKSPACE") THEN DO:
      IF c = lmargin AND r = ptop
      THEN NEXT.
      c = c - 1.
      IF c < lmargin THEN DO:
          c = rmargin.
          r = r - 1.
          IF r < ptop
          THEN r = ptop.
      END.
      PUT SCREEN ROW r COLUMN c " ".
      OVERLAY(comment[r - r-ofst], c - c-ofst) = " ".
      NEXT.
   END.
   IF LASTKEY >= 32 AND LASTKEY <= 126 THEN DO:
      PUT SCREEN ROW r COLUMN c KEYLABEL(LASTKEY).
      OVERLAY(comment[r - r-ofst], c - c-ofst) = KEYLABEL(LASTKEY).
      c = c + 1.
      IF c > rmargin THEN DO:
          c = lmargin.
          r = r + 1.
          IF r > pbot
          THEN r = ptop.
      END.
   END.
END.
DISPLAY comment WITH FRAME x NO-LABELS
   TITLE "Comments Array" 1 COLUMN ROW 09 COLUMN 40.
MESSAGE "Information stored in the comments array.".
```

Notes

- You must use the PUT SCREEN statement to display data when you use the PUT CURSOR statement. You also have to define a variable for the cursor position, and increment it as Progress reads the keys entered by the user if you want the cursor to move as the user types.
- The PUT CURSOR statement displays the cursor until you use the PUT CURSOR OFF statement to stop the display.
- Because a cursor is always displayed in an EDITING phrase, using the PUT CURSOR statement in an EDITING phrase (or if you have not issued a PUT CURSOR OFF statement before the phrase) might cause errors.

See also PUT SCREEN statement

PUT SCREEN statement (Character only)

Displays a character expression at a specified location on a screen, overlaying any other data that might be displayed at that location.

This statement is supported only for backward compatibility.

Note: Does not apply to SpeedScript programming.

Syntax

```
PUT SCREEN
  [ ATTR-SPACE | NO-ATTR-SPACE ]
  [ COLOR color-phrase ]
  [ COLUMN expression ]
  [ ROW expression ]
  expression
```

ATTR-SPACE | NO-ATTR-SPACE

Has no effect; supported only for backward compatibility.

COLOR *color-phrase*

The video attributes you want to use to display an expression. When you display data in the first column of a spacetaking terminal, Progress does not display that data with color. If you are displaying data in a column other than column 1, Progress displays the color attribute in the column prior to the current column (current column minus 1).

```
{   NORMAL
  | INPUT
  | MESSAGES
  | protermcap-attribute
  | dos-hex-attribute
  | { [ BLINK- ]
      [ BRIGHT- ]
      [ fgnd-color ]
      [ bgnd-color ]
    }
  | { [ BLINK- ]
      [ RVV- ]
      [ UNDERLINE- ]
      [ BRIGHT- ]
      [ fgnd-color ]
    }
  | VALUE ( expression )
}
```

For more information, see the COLOR phrase reference entry.

COLUMN *expression*

The column in which you want to display an expression. In the COLUMN option, *expression* is a constant, field name, variable name, or expression whose value is an integer that indicates the column in which you want to display an expression. If you do not use the COLUMN option, PUT SCREEN displays the expression at column 1. If you specify a COLUMN that is outside the screen area, Progress disregards the PUT SCREEN statement.

ROW *expression*

The row in which you want to display an expression. In the ROW option, *expression* is a constant, field name, variable name, or expression whose value is an integer that indicates the row you want to display an expression. If you do not use the ROW option, PUT SCREEN displays the expression at row 1. If you specify a ROW that is outside the screen area, Progress disregards the PUT SCREEN statement.

expression

A constant, field name, variable name, or expression that results in a character string. The character string can contain control characters and can be as long as you want.

Example

The `r-putscr.p` procedure determines whether a customer's current balance is above or below 0. If it is above 0, they have a credit; if it is below 0, they owe money. The label of the balance column is changed based on whether they have a credit or owe money.

r-putscr.p

```
DEFINE VARIABLE paid-owed AS DECIMAL.
DEFINE VARIABLE bal-label AS CHARACTER FORMAT "x(20)".

FOR EACH customer:
    paid-owed = balance.
    IF  paid-owed < 0  /* Customer has a credit */
    THEN DO:
        paid-owed = - paid-owed.
        bal-label = "Customer Credit".
    END.
    ELSE bal-label = "Unpaid balance".
    DISPLAY cust-num name
    paid-owed LABEL " " WITH 1 DOWN.
    IF balance < 0
    THEN PUT SCREEN COLOR MESSAGES ROW 2 COLUMN 34 bal-label.
    ELSE PUT SCREEN ROW 2 COLUMN 34 bal-label.
END.
```

If the customer has a credit (balance < 0) the first PUT SCREEN statement displays the value of bal-label (which is Customer Credit) in the same color as you see system MESSAGES (usually reverse video).

If the customer owes money (balance > 0) the second PUT SCREEN statement displays the value of bal-label (which is Current Balance) in normal display mode.

Notes

- Values displayed by PUT SCREEN are not the same as values that belong to frames. Thus those expressions can be overwritten by other displays or hides. Ensure that values displayed by PUT SCREEN do not overwrite frame fields that are used later for data entry.
- If you use the PUT SCREEN statement in a procedure that runs in batch or background mode, Progress disregards the PUT SCREEN statement.
- The HIDE ALL statement clears the entire screen, including any data displayed by a PUT SCREEN statement.
- The Wyse 75 terminal is spacetaking for some COLOR attributes and non-spacetaking for others. This difference interferes with resetting COLOR MESSAGE (non-spacetaking) back to COLOR NORMAL in a PUT SCREEN statement. If you use WHITE instead of NORMAL whenever you reset color attributes back to normal video attributes, the Wyse 75 behaves like other terminals.
- If you use the PUT SCREEN statement to display data in the message area, the HIDE MESSAGES statement does not necessarily clear that data.

See also COLOR phrase, DISPLAY statement, HIDE statement, PUT statement

PUT statement

Sends the value of one or more expressions to an output destination other than the terminal.

Syntax

```
PUT
  [ STREAM stream ]
  [ UNFORMATTED ]
  [     { expression
            [ FORMAT string ]
            [ { AT | TO } expression ]
        }
      | SKIP [ ( expression ) ]
      | SPACE [ ( expression ) ]
  ] ...
```

```
PUT [ STREAM stream ] CONTROL expression ...
```

`STREAM` *name*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

`UNFORMATTED`

Tells Progress to display each expression in the same format produced by the EXPORT statement, but without quotes.

expression

Specifies a constant, field name, variable name, or expression.

`FORMAT` *`string`*

The format in which you want to display the *`expression`*. If you do not use the FORMAT option, Progress uses the defaults shown in Table 42.

Table 42: Default display formats

Type of expression	Default format
Field	Format from Dictionary.
Variable	Format from variable definition.
Constant character	Length of character string.
Other	Default format for the data type of the expression.

Table 43 shows the default formats for other expressions.

Table 43: Default data type display formats *(1 of 2)*

Data type	Default display format
CHARACTER	x(8)
CLASS[3]	>>>>>>9
DATE	99/99/99
DATETIME	99/99/9999 HH:MM:SS.SSS
DATETIME-TZ	99/99/9999 HH:MM:SS.SSS+HH:MM
DECIMAL	->>,>>9.99
HANDLE[2]	>>>>>>9
INTEGER	->,>>>,>>9
LOGICAL	yes/no
MEMPTR[1]	See the note at the end of the table.

Table 43: Default data type display formats *(2 of 2)*

Data type	Default display format
RAW[1]	See the note at the end of the table.
RECID	>>>>>>9
ROWID[1]	See the note at the end of the table.
WIDGET-HANDLE[2]	>>>>>>9

[1] You cannot display a MEMPTR, RAW, or ROWID value directly. However, you can convert it to a character string representation using the STRING function and display the result. A ROWID value converts to a hexadecimal string, "0x*hexdigits*," where *hexdigits* is any number of characters "0" through "9" and "A" through "F". A MEMPTR or RAW value converts to decimal integer string.

[2] To display a HANDLE or WIDGET-HANDLE, you must first convert it using the INTEGER function and display the result.

[3] To display a CLASS, you must first convert it using the INTEGER or STRING function and display the result.

`AT` *expression*

Specifies the column position where you want to place the output value. If that position has already been used on the current line, PUT skips to the next line and puts the *expression* in the specified column.

`TO` *expression*

Specifies the column position where you want to end the output value being output. If that position has already been used on the current line, PUT skips to the next line and puts the *expression* in the specified column.

`SKIP` **[** `(` *expression* `)` **]**

Specifies the number of new lines you want to output. If you do not use the SKIP option, PUT will not start a new line to the output stream. If you use the SKIP parameter, but do not specify *expression* (or if *expression* is 0), Progress starts a new line only if output is not already positioned at the beginning of a new line.

SPACE [(*expression*)]

Specifies the number of spaces you want to output. Spaces are not placed between items being PUT unless you use the SPACE option.

CONTROL *expression*

The expression specifies a control sequence that you want to send without affecting the current line, page counters, and positions maintained within Progress. Following CONTROL, *expression* can be a character-string expression or a RAW variable. It can include null character constants of the form NULL or NULL(*expression*), where *expression* specifies the number of NULLs to send. See the Notes section for details.

Example

This procedure creates a text file that contains the names of each customer. The names are separated from each other by a slash (/). The entire file consists of one long line.

r-put.p

```
DEFINE STREAM s1.

OUTPUT STREAM s1 TO cus.dat.

FOR EACH customer:
  PUT STREAM s1 name "/".
END.

OUTPUT STREAM s1 CLOSE.
```

Notes

- In the AT, TO, SKIP, and SPACE options, if *expression* is less than or equal to 0, Progress disregards the option.
- The PUT statement never automatically starts a new line. You must use SKIP to explicitly start a new line.

- The PUT statement uses the default display format for the data type of the field or variable you name in the PUT statement. The PUT statement does not overwrite an area that is already used by a previous format when it displays data.

```
DEFINE VARIABLE myname AS CHARACTER FORMAT "x(8)".
DEFINE VARIABLE mynum AS CHARACTER FORMAT "x(8)".

myname = "abc".
mynum = "123".
OUTPUT TO myfile.
PUT myname AT 8 mynum AT 12.
OUTPUT CLOSE.
```

```
abc
123
```

  Use the UNFORMATTED option with the PUT statement to override the format-sensitive display.

- You can use the NULL keyword to output null characters (\0) in a control sequence. For example, the following statements write the control sequence **ESC A** \0 and 20 NULLs to output stream **A**:

```
PUT STREAM A CONTROL "~033A" NULL.
PUT STREAM A CONTROL NULL(20).
```

- You can use the PUT statement with an object reference for a class object instance. The PUT statement implicitly calls the ToString( ) method of the class to convert the specified object reference to a character value before it sends the value to the output destination.

See also DEFINE STREAM statement, DISPLAY statement, EXPORT statement, OUTPUT TO statement, PAGE statement, PUT SCREEN statement

PUT-BITS statement

Uses the bit representation of an integer to set a given number of bits at a given location within another integer. Returns a logical value.

Syntax

```
PUT-BITS( destination , position , numbits ) = expression
```

destination

A Progress integer variable. The statement sets bits in *destination* that correspond to the bits that are on in the source variable, *expression*. It clears bits in the destination variable that are 0 in the source variable. Note that the number of bits set or cleared is limited by the *numbits* parameter, and the location within the destination is determined by the *position* variable.

position

A variable or expression that returns an integer. This parameter designates the position of the lowest-order bit of the bits that are to be interpreted as an integer. Bits are numbered from 1 through the length of an integer; with 1 being the low-order bit. If *position* is greater than the length of an INTEGER or less than 1, Progress generates a runtime error.

numbits

The number of bits to examine when generating the return value. If *position* plus *numbits* is greater than the length of an integer plus 1, Progress generates a runtime error.

expression

A source variable that returns an INTEGER. If the INTEGER cannot be represented in the number of bits specified by numbits, Progress stores the low-order numbits bits of the INTEGER.

See also

GET-BITS function

PUT-BYTE statement

Stores the unsigned 1-byte value of an INTEGER expression at the specified memory location.

Syntax

```
PUT-BYTE ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and has not had its region allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), Progress generates a run-time error.

position

An INTEGER value greater than 0 that indicates the byte position where Progress stores *expression*. If *position* is less than 1, Progress generates a run-time error. For a RAW *destination*, if *position* is greater than the length of *destination*, Progress changes the length of *destination* to *position* and pads the gap with null bytes. For a MEMPTR *destination*, if *position* is greater than the length of *destination*, Progress generates a run-time error.

expression

The INTEGER value of a constant, field, variable, function, or expression. If *expression* is less than 0 or greater than 255, Progress stores the right-most byte value of *expression* in *destination*.

Examples

This procedure finds the name of customer 26, Jack's Jacks, and stores it in the RAW variable r1. The PUT-BYTE statement replaces the first four bytes in the name with the specified character code values. The procedure then writes the values in r1 back into the name field and displays that field. Jack's Jacks becomes Bill's Jacks.

r-rawput.p

```
/* You must connect to a non-PROGRESS demo database to run
   this procedure */

DEFINE VARIABLE r1 AS RAW.

FIND customer WHERE cust-num = 26.
DISPLAY name.
r1 = RAW(name).
PUT-BYTE(r1,1) = ASC('B').
PUT-BYTE(r1,2) = ASC('i').
PUT-BYTE(r1,3) = ASC('l').
PUT-BYTE(r1,4) = ASC('l').

RAW(name) = r1.
DISPLAY name.
```

The following example allocates a MEMPTR region large enough to hold the character string "Bill", terminated by a null byte. It stores the string one byte at a time using the PUT-BYTE statement, and then displays the string directly from the region.

r-mptput.p

```
DEFINE VARIABLE mptr AS MEMPTR.

SET-SIZE(mptr) = LENGTH("Bill") + 1.
PUT-BYTE(mptr,1) = ASC('B').
PUT-BYTE(mptr,2) = ASC('i').
PUT-BYTE(mptr,3) = ASC('l').
PUT-BYTE(mptr,4) = ASC('l').
PUT-BYTE(mptr,5) = 0.

DISPLAY GET-STRING(mptr,1).
```

Note

For more information on accessing DLL routines from Progress, see *OpenEdge Development: Programming Interfaces*.

See also

GET-BYTE function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-BYTES statement

Copies a RAW or MEMPTR variable to the specified location in another RAW or MEMPTR variable.

Syntax

```
PUT-BYTES ( destination , position ) = expression
```

destination

An expression that returns a target RAW or MEMPTR variable. If *destination* is the Unknown value (?), PUT-BYTES does nothing.

position

An integer value greater than 0 that indicates the byte position where you want to put the data. If *position* is less than 1, Progress generates a runtime error.

For a RAW variable, if *position* is greater than the length of *destination*, Progress increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR variable, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, Progress generates a runtime error. If *destination* is a RAW and *position* plus the length of *expression* is greater than 32K, Progress generates a runtime error.

expression

An expression that returns a RAW or MEMPTR variable.

See also

GET-BYTES function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-DOUBLE statement

Stores the 8-byte floating-point value of a DECIMAL expression at the specified memory location.

Syntax

```
PUT-DOUBLE ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and has not had its region allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), Progress generates a run-time error.

position

An INTEGER value greater than 0 that indicates the byte position where Progress stores *expression*. If *position* is less than 1, Progress generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, Progress increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, Progress generates a run-time error.

expression

The DECIMAL value of a constant, field, variable, function, or expression.

Example

For examples of how to use the PUT-DOUBLE statement, see the PUT-BYTE statement reference entry.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from Progress, see *OpenEdge Development: Programming Interfaces*.

See also

GET-DOUBLE function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-FLOAT statement

Stores the 4-byte floating-point value of a DECIMAL expression at the specified memory location.

Syntax

```
PUT-FLOAT ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and has not had its region allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), Progress generates a run-time error.

position

An INTEGER value greater than 0 that indicates the byte position where Progress stores *expression*. If *position* is less than 1, Progress generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, Progress increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, Progress generates a run-time error.

expression

The DECIMAL value of a constant, field, variable, function, or expression.

Example

For examples of how to use the PUT-FLOAT statement, see the PUT-BYTE statement reference entry.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from Progress, see *OpenEdge Development: Programming Interfaces*.

See also

GET-FLOAT function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-KEY-VALUE statement

Adds, modifies, and deletes keys in the current environment.

Note: Does not apply to SpeedScript programming.

Syntax

```
PUT-KEY-VALUE
  { { SECTION section-name
        KEY { key-name | DEFAULT }
        VALUE value
    }
    | { COLOR | FONT } { number | ALL }
  }
  [ NO-ERROR ]
```

SECTION *section-name*

A CHARACTER expression that specifies the name of the section that contains the key of interest.

In initialization files, section names appear in square brackets([]). When you specify a section name in a PUT-KEY-VALUE statement, omit the square brackets.

KEY *key-name*

A CHARACTER expression that specifies the name of the key of interest.

DEFAULT

Tells PUT-KEY-VALUE to use the default key of section *section-name*.

Some applications store data in the registry under the default key of a section. This option lets you modify this data. For an example, see the EXAMPLES section of this entry.

This option applies only to the registry and not to initialization files.

VALUE *value*

The value of the key to write to the environment. *value* must evaluate to a CHARACTER expression of no more than 128 bytes.

COLOR { *number* | ALL }

Updates color definitions in the current environment from the definitions in the internal color table. The *number* parameter is a literal integer that specifies the number of a single color in the current environment whose definition you want to update. The ALL option updates all color definitions in the current environment.

FONT { *number* | ALL }

Updates font definitions in the current environment from the definitions in the internal font table. The *number* parameter is a literal integer that specifies the number of a single font in the current environment whose definition you want to update. The ALL option updates all font definitions in the current environment.

NO-ERROR

Specifies that any errors that occur as a result of the PUT-KEY-VALUE operation are suppressed. After the PUT-KEY-VALUE statement completes, you can check the ERROR-STATUS system handle for information on any errors that might have occurred.

Examples

If the current environment resides in the registry, the following example:

1. Searches in the registry under the current environment for the subkey MYSECTION.
2. Creates MYSECTION if it does not exist.
3. Searches MYSECTION for the subkey MYKEY.
4. Sets MYKEY to the value MYVARIABLE (if MYKEY exists), or adds MYKEY and the value MYVARIBLE (if MYKEY does not exist).

If the current environment resides in an initialization file, the following example:

1. Searches the initialization file for the section MYSECTION.
2. Creates MYSECTION if it does not exist.
3. Searches MYSECTION for the key MYKEY.
4. Sets MYKEY to the value MYVARIABLE (if MYKEY exists), or adds MYKEY and the value MYVARIBLE (if MYKEY does not exist):

```
PUT-KEY-VALUE SECTION "MYSECTION" KEY "MYKEY" VALUE MYVARIABLE
```

If the current environment resides in the registry, the following examples add, directly under the current environment, the value name MYKEY and the value MYVARIABLE.

If the current environment resides in an initialization file, the following examples return an error:

```
PUT-KEY-VALUE SECTION "" KEY "MYKEY" VALUE MYVARIABLE
```

```
PUT-KEY-VALUE SECTION "?" KEY "MYKEY" VALUE MYVARIABLE
```

If the current environment resides in the registry, the following examples:

1. Search in the registry under the current environment for the key MYSECTION.
2. Search MYSECTION for the value name MYKEY.
3. Delete MYKEY and its value.

If the current environment resides in an initialization file, the following examples delete the key MYKEY, including its value, from the section MYSECTION.

```
PUT-KEY-VALUE SECTION "MYSECTION" KEY "MYKEY" VALUE ""
```

```
PUT-KEY-VALUE SECTION "MYSECTION" KEY "MYKEY" VALUE ?
```

If the current environment resides in the registry, the following examples delete the subkey MYSECTION, all values under MYSECTION, all subkeys under MYSECTION, and all values under those subkeys.

If the current environment resides in an initialization file, the following examples remove the section MYSECTION, and all key-value pairs within MYSECTION, from the initialization file:

```
PUT-KEY-VALUE SECTION "MYSECTION " KEY "?" VALUE ?
```

```
PUT-KEY-VALUE SECTION "MYSECTION " KEY "" VALUE ""
```

If the current environment resides in the registry, the following example:

1. Searches the current environment for the subkey MYAPP.
2. Sets the default key under MYAPP to NEWVALUE.

If the current environment resides in an initialization file, the following example returns an error.

```
PUT-KEY-VALUE SECTION "MYAPP" KEY DEFAULT VALUE "NEWVALUE"
```

Notes

- Environments typically consist of sections, each of which contains keys, each of which consists of a name and a value. A typical section name is COLORS. A typical key within this section consists of the name COLOR7 and the value 255,255,0. This key attaches the name COLOR7 to color value 255,255,0 (a color specification that uses the red-green-blue color-naming scheme).

  The current environment might be the registry or an initialization file. The registry consists of sections called keys and subkeys arranged in a hierarchy. Keys and subkeys contain value entries, each of which consists of a value name and value data. Initialization files, by contrast, consist of a single level of sections. Sections contain entries, each of which consists of a name, an equal sign (=), and a value.

  For more information on environments, see the chapter on colors and fonts in *OpenEdge Development: Programming Interfaces*.

- The current environment is one of the following:
  - The default environment.
  - An environment that a startup parameter specified (This environment is called the *startup environment*).
  - An environment that a LOAD statement loaded and that the most recent USE statement made current.
- If you UNLOAD the current environment, a subsequent PUT-KEY-VALUE writes to the startup environment.
- To remove a key-value pair from an environment, set *key-name* to the name of the key and *value* to the Unknown value (?).

- To remove a section, including all its key-value pairs, from an environment, set *section-name* to the name of the section and *key-name* to the Unknown value (?).
- To change the definitions in the internal color table, use one of the following techniques:
  - To display a dialog box that lets the user change the color definitions, use the SYSTEM-DIALOG-COLOR statement.
  - To change the color definitions directly from the 4GL, use the attributes and methods of the COLOR-TABLE handle.

Note: The COLOR option of the PUT-KEY-VALUE statement does not change the definitions in the internal color table. This option merely moves some or all of those definitions to the current environment.

- To change the definitions in the internal font table, use one of the following techniques:
  - To display a dialog box that lets the user change the font definitions, use the SYSTEM-DIALOG-FONT statement.
  - To change the font definitions directly from the 4GL, use the attributes and methods of the FONT-TABLE handle.

Note: The FONT option of the PUT-KEY-VALUE statement does not change the definitions in the internal font table. This option merely moves some or all of those definitions to the current environment.

- For more information on colors and fonts, see the chapter on colors and fonts in *OpenEdge Development: Programming Interfaces*.

See also COLOR-TABLE system handle, FONT-TABLE system handle, GET-KEY-VALUE statement, LOAD statement, SYSTEM-DIALOG COLOR statement, SYSTEM-DIALOG FONT statement, UNLOAD statement, USE statement

PUT-LONG statement

Stores the signed 32-bit value of an INTEGER expression at the specified memory location.

Syntax

```
PUT-LONG ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region in not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), Progress generates a run-time error.

position

An INTEGER value greater than 0 that indicates the byte position where Progress stores *expression*. If *position* is less than 1, Progress generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, Progress increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, Progress generates a run-time error.

expression

The INTEGER value of a constant, field, variable, function, or expression.

Example

For examples of how to use the PUT-LONG statement, see the PUT-BYTE statement reference entry.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from Progress, see *OpenEdge Development: Programming Interfaces*.

See also

GET-LONG function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-SHORT statement

Stores the signed 16-bit value of an INTEGER expression at the specified memory location.

Syntax

```
PUT-SHORT ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region is not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), Progress generates a run-time error.

position

An INTEGER value greater than 0 that indicates the byte position where Progress stores *expression*. If *position* is less than 1, Progress generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, Progress increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, Progress generates a run-time error.

expression

The INTEGER value of a constant, field, variable, function, or expression.

Example

For examples of how to use the PUT-SHORT statement, see the PUT-BYTE statement reference entry.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from Progress, see *OpenEdge Development: Programming Interfaces*.

See also

GET-SHORT function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-STRING statement

Stores the null-terminated value of a CHARACTER or LONGCHAR expression at the specified memory location. If *numbytes* is specified, PUT-STRING will copy the requested number of bytes from the variable, regardless of whether there are embedded nulls. In this case PUT-STRING will not put a terminating null into the MEMPTR unless the last byte copied happens to be a null.

Syntax

```
PUT-STRING ( destination , position , [ numbytes ] ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region is not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), Progress generates a run-time error.

position

An INTEGER value greater than 0 that indicates the byte position where Progress stores *expression*. If *position* is less than 1, Progress generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, Progress increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, Progress generates a run-time error.

numbytes

An integer value greater than 0 that indicates how many bytes to copy from *expression.* If *position* plus *numbytes* is greater than the length of *destination*, Progress generates a runtime error.

expression

An expression (a constant, field name, variable name, or expression) whose value is a CHARACTER or LONGCHAR. Progress converts a LONGCHAR value to `-cpinternal` before it stores the value.

Example For examples of how to use the PUT-STRING statement, see the PUT-BYTE statement reference entry.

Note For more information on accessing DLL and UNIX shared library routines from Progress, see *OpenEdge Development: Programming Interfaces*.

See also GET-STRING function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

PUT-UNSIGNED-SHORT statement

Stores the unsigned 16-bit value of an INTEGER expression at the specified memory location.

Syntax

```
PUT-UNSIGNED-SHORT ( destination , position ) = expression
```

destination

A variable of type RAW or MEMPTR. If *destination* is the Unknown value (?), it remains the Unknown value (?). If *destination* is a MEMPTR and its region is not allocated (by a SET-SIZE statement or by a Windows dynamic link library (DLL) or UNIX shared library routine), Progress generates a run-time error.

position

An INTEGER value greater than 0 that indicates the byte position where Progress stores *expression*. If *position* is less than 1, Progress generates a run-time error.

For a RAW *destination*, if *position* is greater than the length of *destination*, Progress increases the length of *destination* to *position* plus the remaining bytes needed to store *expression*. The gap between the original *destination* length and *position* is padded with null bytes.

For a MEMPTR *destination*, if *position* is greater than the length of *destination* or does not leave sufficient room to store *expression*, Progress generates a run-time error.

expression

The INTEGER value of a constant, field, variable, function, or expression.

Notes

- This statement supports byte-swapping only if *destination* is a MEMPTR data type. The statement will first examine the byte-order setting of the MEMPTR and then swap the bytes appropriately while putting the data into the MEMPTR memory.
- For more information on accessing DLL routines from Progress, see *OpenEdge Development: Programming Interfaces*.

See also

GET-UNSIGNED-SHORT function, LENGTH function, LENGTH statement, RAW function, RAW statement, SET-SIZE statement

QUERY-OFF-END function

Returns a logical value indicating whether the specified query is positioned at the end of its result list (either before the first record or after the last record).

Syntax

```
QUERY-OFF-END ( query-name )
```

query-name

A character expression that evaluates to the name of a currently open query. If *query-name* does not resolve to the name of a query, or if the query is not open, then the function returns the Unknown value (?).

Note: Searching for a query using a handle is more efficient than a character expression. Progress resolves a character expression at runtime by searching in the current routine for a static query with that name. If not found, Progress searches the enclosing main procedure. If still not found, Progress searches up through the calling programs of the current routine, and their main procedures. Since a handle uniquely identifies a query, no such search is required. Use the query object handle's QUERY-OFF-END attribute to avoid a runtime search.

Example

The following example uses the QUERY-OFF-END function to determine when to leave the REPEAT loop:

r-qoff.p

```
OPEN QUERY cust-query FOR EACH customer.

REPEAT:
   GET NEXT cust-query.

   IF QUERY-OFF-END("cust-query")
   THEN LEAVE.

   DISPLAY cust-num name.
END.
```

When you run this procedure, all customer numbers and names are displayed. After the last record is displayed, the loop iterates and the GET NEXT statement reads beyond the last record. At this point QUERY-OFF-END returns TRUE and Progress exits the loop.

Note To test whether a GET statement read beyond the last (or first) record pass a buffer to the AVAILABLE function. The QUERY-OFF-END function serves the same purpose, but does not require a specific buffer; it requires only a query name.

See also CLOSE QUERY statement, CURRENT-RESULT-ROW function, DEFINE BROWSE statement, DEFINE QUERY statement, GET statement, NUM-RESULTS function, OPEN QUERY statement, QUERY-OFF-END attribute, REPOSITION statement

QUERY-TUNING phrase

Allows programmatic control over the execution of a query in a DataServer application. This phrase is available for the DataServers; it is not available for queries of OpenEdge databases.

Syntax

```
QUERY-TUNING
  (
    { [ ARRAY-MESSAGE | NO-ARRAY-MESSAGE ]
      [ BIND-WHERE | NO-BIND-WHERE ]
      [ CACHE-SIZE integer ]
      [ DEBUG { SQL | EXTENDED diag-option } | NO-DEBUG ]
      [ INDEX-HINT | NO-INDEX-HINT ]
      [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
      [ LOOKAHEAD | NO-LOOKAHEAD ]
      [ ORDERED-JOIN ]
      [ REVERSE-FROM ]
      [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
    }
  )
```

The following descriptions are general. For more detailed information, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

ARRAY-MESSAGE | NO-ARRAY-MESSAGE

Specifies whether the DataServer sends multiple result rows in a single logical network message.

The default is ARRAY-MESSAGE.

```
BIND-WHERE | NO-BIND-WHERE
```

This option is available only for the DataServer for ORACLE.

Specifies whether the DataServer uses ORACLE bind variables or literals in WHERE clauses. If you use NO-BIND-WHERE, the DataServer uses literals. Bind variables can improve performance, but ORACLE produces some unexpected results for some data types.

The default is BIND-WHERE.

```
CACHE-SIZE integer [ ROW | BYTE ]
```

Specifies the maximum cache size the DataServer can use when fetching records for a lookahead or standard cursor. You can optionally specify the size of the cache information in either bytes or records. The following values are for ORACLE.

The default is 1024 for standard cursors and 8192 for lookahead cursors.

If you use the byte option, the byte maximum is 65535 bytes and the byte minimum specifies the number of bytes contained in a single record. For joins, you must specify the number of bytes contained in two records.

If you use the row option, the row maximum equals the maximum number of records that can be fit in 65535 bytes. The row minimum is 1 row for a single table and 1 rows for a join.

The default is 30000.

```
{ DEBUG { SQL | EXTENDED diag-option } } | NO-DEBUG
```

Specifies whether the DataServer should print debugging information for the query to the `dataserv.lg` file.

The SQL option prints the SQL executed by the DataServer against the non-OpenEdge DBMS. The extended option prints additional information, such as cursor statistics. The information you get when you use the EXTENDED option can be helpful in setting your parameters.

The default is NO-DEBUG.

EXTENDED *diag-option*

The syntax for the diagnostic options is as follows:

```
EXTENDED CURSOR | DATA-BIND | PERFORMANCE | VERBOSE
```

For more information, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

HINT

This option is only available for the DataServer for ORACLE.

Specifies the ORACLE hint syntax that the DataServer passes directly to the ORACLE DBMS as part of the query. This allows you to control which hints are passed as opposed to the index hints that the DataServer passes when appropriate.

INDEX-HINT | NO-INDEX-HINT

This option is available only for the DataServer for ORACLE.

Specifies whether the DataServer provides index hints to the ORACLE DBMS. INDEX-HINT places index hints in the generated SQL; NOINDEX-HINT prevents the use of index hints.

The default is INDEX-HINT.

JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB

Specifies whether the non-OpenEdge DBMS can perform joins when possible, which usually improves performance.

The default is JOIN-BY-SQLDB.

LOOKAHEAD | NO-LOOKAHEAD

Specifies whether the DataServer uses lookahead or standard cursors. Lookahead cursors fetch as many records as can fit into the allocated cache, which reduces the number of database accesses and improves performance.

The default is LOOKAHEAD, except with statements that use an EXCLUSIVE lock.

ORDERED-JOIN

Specifies that the DataServer embed the ORDERED hint syntax in the SQL it generates. Applies to ORACLE only.

REVERSE-FROM

Specifies that tables are joined in the reverse order in which they appear in the FROM clause. Applies to ORACLE only.

SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION

Creates a new connection for each cursor that the DataServer opens. Applies to the OpenEdge DataServer for ODBC only.

Example

The following code fragment illustrates a QUERY-TUNING phrase in a FOR EACH statement. In this example, the DataServer uses lookahead cursors with a cache size of 32K and records debugging information:

```
FOR EACH customer, EACH order OF customer WHERE ord-num > 20
  BY cust-num
  QUERY-TUNING(LOOKAHEAD CACHE-SIZE 32768 DEBUG EXTENDED)
  TRANSACTION:
```

Note

For the DataServer for ORACLE, all options of the QUERY-TUNING phrase are effective at both compile and run time, except INDEX-HINT, NO-INDEX-HINT, JOIN-BY-SQLDB, and NO-JOIN-BY-SQLDB, which are only effective at compile time.

For more information on the QUERY-TUNING phrase, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

See also

DO statement, FOR statement, OPEN QUERY statement, REPEAT statement

QUIT statement

Raises the QUIT condition. By default, this exits from Progress and returns to the operating system. When QUIT is executed from within a procedure running on an AppServer, it terminates the OpenEdge session running on the AppServer, causing the AppServer server to shut down and returns to the OpenEdge client session from which it was spawned.

Note: Does not apply to SpeedScript programming.

Syntax

```
QUIT
```

Example

This procedure displays a menu. If you choose the last menu item, Exit Progress, the procedure processes the QUIT statement.

r-quit1.p

```
DEFINE SUB-MENU cusmaint1
    MENU-ITEM crecust LABEL "Create New Customer"
    MENU-ITEM chgcust LABEL "Chan&ge Existing Customer"
    MENU-ITEM delcust LABEL "Delete Customer"
    MENU-ITEM prtcust LABEL "Print Customer List"
    MENU-ITEM extcust LABEL "E&xit PROGRESS".

DEFINE MENU mainbar MENUBAR
    SUB-MENU cusmaint1 LABEL "Customer".

ON CHOOSE OF MENU-ITEM crecust
   RUN newcust.p.

ON CHOOSE OF MENU-ITEM chgcust
   RUN chgcust.p.

ON CHOOSE OF MENU-ITEM delcust
   RUN delcust.p.

ON CHOOSE OF MENU-ITEM prtcust
   RUN prncust.p.

ON CHOOSE OF MENU-ITEM extcust
   QUIT.

CURRENT-WINDOW:MENUBAR = MENU mainbar:HANDLE.
CURRENT-WINDOW:VISIBLE = TRUE.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- To modify the QUIT statement, by add the ON QUIT phrase to a block.
- If QUIT is executed during a transaction, Progress commits the transaction before exiting.

See also

ON QUIT phrase, STOP statement

QUOTER function

Converts the specified data type to CHARACTER and encloses the results in quotes when necessary.

The QUOTER function is intended for use in QUERY-PREPARE where a character predicate must be created from a concatenated list of string variables to form a WHERE clause. In order to process variables, screen values, and input values so that they are suitable for a query WHERE clause, it is often necessary to enclose them in quotes. For example, European-format decimals and character variables must always be enclosed in quotes. You can use the Quoter function to meet that requirement.

Syntax

```
QUOTER (<expression> [, <quote-char> [ , <null-string> ]])
```

expression

An *expression* in the data type that you want to convert to character and enclose with quotes.

quote-char

Either a single or double quote, enclosed in the opposite: """ or "'". The default is double quote. Passing ? for this argument results in double quotes.

null-string

The string you want for an unknown value: the word NULL or "" for example. The default is an unquoted question mark, which is the Unknown value (?).

For example, the following:

```
 DEFINE VARIABLE mychar As CHARACTER INITIAL "Lift Line Skiing".
     …..
     qhandle:QUERY-PREPARE( "FOR EACH customer WHERE name = "  +
QUOTER(mychar))
```

Would produce this prepare-string:

```
FOR EACH customer WHERE name = "Lift Line Skiing".
```

Notes

- To address the situation where an Unknown value (?) in a list of concatenated strings could cause the entire string to be unknown and the QUERY-PREPARE to fail, the QUOTER function does **not** return the Unknown value (?) if the *<expression>* argument is unknown. Instead, it returns a known character value consisting of an UNQUOTED question-mark, by default, or the 3rd argument, if it is present.
- Also, in this situation, a quoted question-mark is not used because it is interpreted as string data in a WHERE clause. After the concatenation is complete, Progress supplies a normal question mark.

  For example, the following:

```
 DEFINE VARIABLE mychar As CHARACTER.
     …..
     mychar = ?.
     qhandle:QUERY-PREPARE( "FOR EACH customer WHERE name = "  +
QUOTER(mychar)).
```

  Would produce this prepare-string:

```
FOR EACH customer WHERE name = ?.
```

  However, giving the 3rd parameter as `"NULL"` produces `NULL` rather than `?`.

- For noncharacter data types, if *<expression>* is of type DECIMAL, INTEGER, DATE , and so on, the following occurs:
  - The *<expression>* is converted to character and enclosed in quotes. The conversion is similar to the EXPORT format. DATE types, however, always have the 4-digit year.
  - Data types with no DISPLAY format like MEMPTR and LVARBINARY return the Unknown value (?).
  - If a data type is of type RAW, it is converted to base 64.

For example, the following:

```
 DEFINE VARIABLE mydec As DECIMAL INITIAL 12.34.
     …..
     qhandle:QUERY-PREPARE( "FOR EACH customer WHERE balance = "  +
QUOTER(mydec)).
```

Would produce this prepare-string:

```
FOR EACH customer WHERE balance = "12.34".
```

This is especially important for European format decimals that look like 12,34 and would not compile in the above statement unless they are enclosed in quotes.

- If *<expression>* is of data type CHARACTER, internal quotes are doubled. If the first and last byte are already quotes, then it is assumed that the quoting has already been done, and no further quotes are applied.
- You can use the QUOTER function with an object reference for a class object instance to obtain a unique object identifier within the session as a quoted character string.

R-INDEX function

Returns an integer that indicates the position of the target string within the source string. In contrast to the INDEX function, R-INDEX performs the search from right to left.

Syntax

```
R-INDEX ( source , target [ , starting ] )
```

source

A character expression. This can be a constant, field name, variable name, or expression that results in a character value.

target

A character expression whose position you want to locate in *source*. If *target* does not exist within *source*, R-INDEX returns 0.

If a *starting* parameter is not specified, then the search for the *target* pattern begins at the right-most character. Even though the search is started from the right, the *target* position is calculated from the left. For example, this code returns a 3 rather than a 2:

```
R-INDEX("abcd" , "c")
```

starting

An integer that specifies the begin point for the search. The search is right-to-left and starts from the starting point. For example, this statement returns 1 R-INDEX("abcdefabcdef","abc",6).

Examples

This procedure prompts you to enter a character string and a pattern to match against the string. It then displays the starting position of the string where the pattern was found.

r-rindex.p

```
DEFINE VARIABLE rindx AS INTEGER.
DEFINE VARIABLE source AS CHARACTER FORMAT "X(45)".
DEFINE VARIABLE target AS CHARACTER FORMAT "X(45)".

REPEAT:
  PROMPT-FOR source
    LABEL "Enter a character string to do pattern matching:"
      WITH FRAME s1 CENTERED.
  PROMPT-FOR target
    LABEL "Enter a pattern to match in the string:"
      WITH FRAME t1 CENTERED.
  rindx = R-INDEX(INPUT source, INPUT target).
  IF rindx < > 0 THEN DO:
    DISPLAY "The target pattern:" INPUT target NO-LABEL
        "last appears in position" rindx NO-LABEL SKIP
        WITH FRAME r1 ROW 12 CENTERED.
    DISPLAY "in the source string:" INPUT source NO-LABEL
        WITH FRAME r1 ROW 12 CENTERED.
    HIDE FRAME r1.
  END.
  IF rindx = 0 THEN DO:
    DISPLAY "The target pattern:" INPUT target NO-LABEL
      "could not be found" SKIP
      WITH FRAME r2 ROW 12 CENTERED.
    DISPLAY "in the source string:" INPUT source NO-LABEL
      WITH FRAME r2 ROW 12 CENTERED.
    HIDE FRAME r2.
  END.
END.
```

This example also uses a *starting* value:

r-rndex.p

```
DEFINE VARIABLE mark AS INTEGER.
DEFINE VARIABLE line-width AS INTEGER.
DEFINE VARIABLE paragraph AS CHARACTER.

paragraph = "The course centers around an existing small "
          + "application that you modify to improve perfo"
          + "rmance. Our highly-qualified instructors dem"
          + "onstrate proven analysis and coding techniqu"
          + "es and provide tips for making the most of y"
          + "our PROGRESS code. You are encouraged to bri"
          + "ng your own application problems to class an"
          + "d actively participate in class discussions "
          + "and hands-on lab exercises.".

SET line-width LABEL "Justify with how many character wide?"
   VALIDATE(line-width >= 20 AND line-width <= 70,
            "Must be between 20 and 70 for this example.")
   WITH SIDE-LABELS FRAME ask.FORM
   paragraph FORMAT "x(72)"
   WITH DOWN NO-LABELS USE-TEXT.

DISPLAY "L" + FILL("-", line-width - 2) + "R" @ paragraph.
DOWN.

DO WHILE LENGTH(paragraph) > line-width:
   mark = R-INDEX(paragraph, " ", line-width).
   DISPLAY SUBSTR(paragraph, 1, mark) @ paragraph.
   DOWN.
   paragraph = SUBSTR(paragraph, mark + 1).
END.
IF paragraph <> ""
THEN DISPLAY paragraph.
```

Notes

- If either operand is case sensitive, then the R-INDEX function is also case sensitive.
- If either the *source* string or *target* pattern is null, the result is 0.
- The R-INDEX function is double-byte enabled. You can specify *target* and *source* strings for the R-INDEX function that contain double-byte characters.

See also INDEX function, LOOKUP function

RADIO-SET phrase

Describes a radio set representation for a field or variable. The RADIO-SET phrase is an option of the VIEW-AS phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
RADIO-SET
  [ HORIZONTAL [ EXPAND ] | VERTICAL ]
  [ size-phrase ]
  RADIO-BUTTONS label, value [ , label , value ] ...
  [ TOOLTIP tooltip ]
```

HORIZONTAL

Specifies that the radio buttons are aligned horizontally. Vertical alignment is the default.

VERTICAL

Specifies that the radio buttons are aligned vertically. Because this is the default alignment, you do not have to supply this attribute.

EXPAND

Pads all button labels to be the width of the widest radio button label. This ensures that the buttons are evenly spaced. Use this option only in conjunction with the HORIZONTAL option. If you do not specify this option, the individual radio buttons are spaced evenly if the lengths of the labels vary.

size-phrase

Specifies the outside dimensions of the radio-set widget. This is the syntax for size-phrase:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For further information, see the SIZE phrase reference entry.

RADIO-BUTTONS *label*, *value* [, *label*, *value*] . . .

A list of radio buttons whose selections are mutually exclusive. Each button is composed of a label and value pair. The *label* is a character string that is the label for the radio button. The *value* is the value to be assigned to the field or variable if the radio button is selected; *value* must be a valid value for the field or variable.

You can designate a character within each *label* as a navigation mnemonic in Windows. Indicate the character by preceding it with an ampersand (&). When the radio set is displayed, the mnemonic is underlined. The user can choose to the specific button by pressing **ALT** and the underlined letter.

Note: If two or more buttons of a radio set use the same label, the Progress 4GL uses only the value of the first button.

TOOLTIP *tooltip*

Allows you to define a help text message for a text field or text variable. Progress automatically displays this text when the user pauses the mouse button over a text field or text variable for which a tooltip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the tooltip is removed. No tooltip is the default. The TOOLTIP option is supported in Windows only.

Example

This procedure displays a radio set that consists of three radio buttons and prompts the user to select one of the buttons. When the user selects the button, the program displays the text “This event occurred on” and the date value of selected button.

r-radio1.p

```
DEFINE VARIABLE hist-date AS DATE FORMAT "99/99/9999" INITIAL 07/04/1776
             VIEW-AS RADIO-SET
             RADIO-BUTTONS "Declaration of Independence", 07/04/1776,
                           "Lee Surrenders to Grant", 04/07/1865,
                           "Man Walks on Moon", 07/11/1969.

FORM
   hist-date
  WITH FRAME main-frame NO-LABELS TITLE "Dates in US History".ON VALUE-CHANGED
OF hist-date
   DO:
    ASSIGN hist-date.
    DISPLAY "This event occurred on " + STRING(hist-date) FORMAT "x(60)"
        WITH FRAME main-frame.
   END.
ENABLE hist-date WITH FRAME main-frame.APPLY "VALUE-CHANGED" TO hist-date.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

See also

VIEW-AS phrase

RANDOM function

Returns a random integer between two integers (inclusive).

Note: This function returns a number from a **pseudorandom** sequence of numbers rather than a truly random sequence.

The Alternate Random Number Generator (-rand) parameter determines whether the same sequence of random numbers is generated for each session. For information on this parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

Syntax

```
RANDOM ( low , high )
```

low

An integer expression that is the lower of the two expressions you are supplying to the RANDOM function.

high

An integer expression that is the higher of the two expressions you are supplying to the RANDOM function.

Example

Often when you set up a database for testing purposes, you want to generate many records without actually keying in data for each record. The `r-random.p` procedure generates 10 order records and a random number of order-lines for each order record.

r-random.p

```
DEFINE VARIABLE onum AS INTEGER.
DEFINE VARIABLE olnum AS INTEGER.

DO onum = 1 TO 10 TRANSACTION:
    CREATE order.
    order.order-num = onum.
    order.order-date = TODAY.
    DO olnum = 1 TO RANDOM(1,9):
        CREATE order-line.
        order-line.line-num = olnum.
        order-line.item-num = olnum.
    END.
END.
```

RAW function (ORACLE only)

Extracts bytes from a field.

Syntax

```
RAW ( field [ , position [ , length ] ] )
```

field

Any field from which you want to extract bytes.

position

An integer expression that indicates the position of the first byte you want to extract from *field*. The default value of *position* is 1.

length

An integer expression that indicates the number of bytes you want to extract from *field*. If you do not use the *length* argument, RAW uses *field* from *position* to end.

Example

This procedure extracts bytes from the name field of the first customer, starting at byte 8, and writes 4 bytes to the variable r1:

r-rawfct.p

```
/*You must connect to a non-PROGRESS demo database to run this procedure*/

DEFINE VARIABLE r1 AS RAW.

FIND FIRST customer.
r1 = RAW(name,8,4).
```

Notes

- If *position* is less a 1, or *length* is less than 0, Progress returns a run-time error.
- If (*position* + *length* -1) is greater than the length of the field from which you are extracting the bytes, Progress returns a run-time error.

See also

GET-BYTE function, LENGTH statement, PUT-BYTE statement, RAW statement

RAW statement (ORACLE only)

Writes bytes to a field.

Syntax

```
RAW ( field [ , position [ , length ] ] ) = expression
```

field

The field in which you want to store *expression*.

position

An integer expression that indicates the position in *field* where you want to store *expression*. The default for *position* is 1.

length

An integer expression that indicates the number of positions you want to replace in *field*. If you do not use the *length* argument, RAW puts *expression* into *field* from *position* to end. Progress treats variable-length fields and fixed-length fields differently. See the Notes section for more information.

expression

A function or variable name that returns data and results in the bytes that you want to store in *field*.

Notes

- In a variable length field, if (*position* + *length* -1) is greater than the length of *field*, Progress pads the field with nulls before it performs the replacement.
- In a fixed length field, if (*position* + *length* -1) is greater than the length of *field*, Progress returns a run-time error. If (*position* + *length* -1) is less then the length of *field*, Progress pads the field with nulls so that it remains the same size.
- If *position*, *length*, or *expression* is equal to the Unknown value (?), then *field* becomes the Unknown value (?).
- If *position* is less than 1, or *length* is less than 0, Progress generates a run-time error.

See also

GET-BYTE function, LENGTH function, LENGTH statement, PUT-BYTE statement, RAW function

RAW-TRANSFER statement

Copies a record wholesale from a source to a target.

Syntax

```
RAW-TRANSFER
  {     [ BUFFER ]  buffer     TO  [ FIELD ]   raw-field
    |   [ FIELD ]   raw-field  TO  [ BUFFER ]  buffer
    |   [ BUFFER ]  buffer     TO  [ BUFFER ]  buffer
  }
  [ NO-ERROR ]
```

BUFFER

Specifies a parameter is a buffer.

buffer

A source or target database record.

Note: If the source buffer contains only a partial field list, RAW-TRANSFER fails.

FIELD

Specifies a parameter is a raw-field.

raw-field

A source or target data field of type RAW.

NO-ERROR

Suppresses Progress's run-time error behavior and stores information on run-time errors, if any, in the ERROR-STATUS system handle.

Example

The following Progress 4GL example performs a RAW-TRANSFER of a newly created Customer record to the Record field of Replication-Log table:

```
TRIGGER PROCEDURE FOR REPLICATION-CREATE OF Customer.
CREATE Replication-Log.
ASSIGN
  Replication-Log.Taskid = DBTASKID(LDBNAME(BUFFER Replication-Log))
  Replication-Log.Table = 'Customer'
  Replication-Log.Action = 'CREATE'.
RAW-TRANSFER Customer TO Replication-Log.Record.
```

For more information on database replication, see *OpenEdge Data Management: Database Administration*.

Notes

- The RAW-TRANSFER statement has several variations:
  - The “buffer to raw-field” variation copies the entire record from the buffer to the raw field, prepending information on the source schema to the raw field.
  - The “raw-field to buffer” variation first checks that the source schema information prepended to the raw field matches the schema of the buffer. Then it creates a target record, if necessary. Finally it updates each key field in the new record using values from the raw field, which forces indexing to occur.
  - The “buffer to buffer” variation is the same as the “raw-field to buffer” variation, except that the source is a record in another buffer.
- The RAW-TRANSFER statement respects database triggers.
- You can marshal an OpenEdge database record so that it can be sent across sockets by using the RAW-TRANSFER statement to put the record into a RAW variable and then copying the RAW variable to a MEMPTR that is being written to a socket. Use the PUT-BYTES function to do this. You can unmarshal database records by using the GET-BYTES function and then RAW-TRANSFER.

- At run time, the RAW-TRANSFER statement:
  - Checks that the signatures of the source data and the target data match.
  - Compares source and target code page ids, and (if they are present and different) translates the source's character data, writing any warnings to the database log file and raising any error conditions.
  - Creates the target record, if none exists, and runs all appropriate CREATE triggers (unless the DISABLE TRIGGERS FOR LOAD option is active for the target).
  - Registers changes in key fields with the index manager by updating each key field in the target when it differs from the source.
  - Copies all data from the source record to the target record.
  - Executes ASSIGN triggers for any modified fields (unless the DISABLE TRIGGERS FOR LOAD option is active for the target).
- When using the RAW-TRANSFER statement to copy a record that contains a BLOB or CLOB field, Progress skips the BLOB or CLOB field and stores the Unknown value (?) in the BLOB or CLOB field of the target record.

See also DISABLE TRIGGERS statement, LDBNAME function, RAW-TRANSFER( ) method, RECORD-LENGTH function

READKEY statement

Reads one keystroke from an input source and sets the value of LASTKEY to the keycode of that keystroke. Use the READKEY statement when you want to look at each keystroke a user makes and take some action based on that keystroke.

Note: Does not apply to SpeedScript programming.

Syntax

```
READKEY [ STREAM stream ] [ PAUSE n ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement.

PAUSE *n*

The READKEY statement waits up to *n* seconds for a keystroke. If you do not press a key during that amount of time, READKEY ends, and sets the value in LASTKEY to -1.

PAUSE 0 causes READKEY to immediately return a value. If no character is available, READKEY sets the value of LASTKEY to -1. Use this form of READKEY to do polling through UNIX pipes or terminal ports.

Example

In the following procedure, when the user presses a key, the READKEY statement reads the keystroke and stores the character code value of that key (the key code) as the value of LASTKEY. The CHR function converts the character code value into a character value. If the character value is a Y, Progress deletes the customer. KEYFUNCTION determines the function of the LASTKEY. If that function is END-ERROR, Progress exits the block, ending the procedure.

r-readky.p

```
FOR EACH customer:
  DISPLAY cust-num name address city st WITH 1 DOWN.
  MESSAGE "If you want to delete this customer, press Y".
  MESSAGE "Otherwise, press any other key.".
  READKEY.
  IF CHR(LASTKEY) = "Y"
  THEN DELETE customer.
  ELSE IF KEYFUNCTION(LASTKEY) = "END-ERROR"
  THEN LEAVE.
END.
```

Notes

- If you use READKEY, it intercepts any input from the user. Thus no widgets receive the input. To pass the input to a widget, you must use the APPLY statement.
- The READKEY function is double-byte enabled. The READKEY function returns values only after the input method places the data in the keyboard buffer. It returns the key code of the most recent key sequence returned from the keyboard buffer. A key sequence is the set of keystrokes necessary to generate one character or function key event in Progress.
- If the current input source is a file, then READKEY reads the next character from that file and returns the value of that character (1 to 255) to LASTKEY. READKEY does not translate periods (.) in the file into the ENDKEY value. It does translate end of line into RETURN (13), but it cannot read any special keys, such as function keys.

  When Progress reaches the end of the file, it sets the value of LASTKEY to -2, but does not close the input file. At that point, an APPLY LASTKEY (same as APPLY -2) raises the ENDKEY condition.
- If the current input source is a UNIX pipe, any timer you set with the PAUSE option might expire before READKEY can read a character. If so, LASTKEY is set to -1.

- If the last key typed is an invalid character sequence, READKEY sets the value of LASTKEY to -1.
- On UNIX System V machines, if input is coming from a pipe, READKEY PAUSE 0 is treated as READKEY PAUSE 1. There is always a 1 second wait for input.
- READKEY counts to determine whether an UNDO, RETRY should be treated as UNDO, NEXT, and whether UNDO, NEXT should be treated as UNDO, LEAVE . This presents infinite loops.
- For more information on monitoring keystrokes, see *OpenEdge Development: Programming Interfaces*.

See also DEFINE STREAM statement, LASTKEY function

RECID function

Returns the unique internal identifier of the database record currently associated with the record buffer you name. This internal identifier has the data type RECID, a four-byte value that is supported by OpenEdge databases and some non-OpenEdge DataServers.

Note: Supported mainly for backward compatibility. For most applications, use the ROWID function, instead. For more information, see the ROWID function reference entry.

Syntax

```
RECID ( record )
```

record

The name of the record whose RECID you want.

To use the RECID function with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

Example

You might decide that you do not want to lock a record until the user starts to update that record. In the example procedure, the FIND statement reads a customer record without locking the record. The RECID function puts the internal database identifier of that record in the crecid variable. If the user decides to update the credit-limit field, the procedure finds the record again using the value in crecid. The second FIND statement reads the record again, this time placing an EXCLUSIVE-LOCK on it. Because the record is first found with NO-LOCK, it is possible for the record to be updated by another user after the first FIND and before the second.

r-recid.p

```
DEFINE VARIABLE response AS LOGICAL.
DEFINE VARIABLE crecid AS RECID.

REPEAT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING cust-num NO-LOCK.
    crecid = RECID(customer).
    DISPLAY name .
    response = YES.
    UPDATE response LABEL "Update credit-limit ?".
    IF response THEN DO:
        FIND customer WHERE RECID(customer) = crecid EXCLUSIVE-LOCK.
        UPDATE credit-limit.
    END.
END.
```

Notes

- Use the RECID function to rapidly retrieve a previously identified record, even if that record has no unique index.
- If you want a called procedure to use the same record as a calling procedure, use the RECID function to ensure that you are retrieving the same record. Use a SHARED variable to communicate the RECID of a record from one procedure to another. The second procedure can then find the same record. This is an alternative to using shared buffers.
- Avoid storing RECID values in database fields because those RECIDs will change if you dump and reload the database.

- You do not have to explicitly check to see whether a record is AVAILABLE before using the RECID function. The RECID function returns the Unknown value (?) if a record cannot be accessed.

  This example displays a RECID only when a record can be accessed:

```
DISPLAY
  (IF AVAILABLE customer
  THEN RECID(customer) ELSE ?).
```

  Directly reference RECID even if a record cannot be found:

```
FOR EACH customer:
  DISPLAY cust-num.
END.

DISPLAY RECID(customer).
```

See also DEFINE BUFFER statement, DEFINE VARIABLE statement, Record phrase, ROWID function

Record phrase

Identifies the record or records you want to verify using the CAN-FIND function, retrieve with a FIND statement, query with a FOR statement or OPEN QUERY statement, or preselect in a DO or REPEAT block.

The Record phrase syntax describes three kinds of information:

- Qualifies the record(s) to access in the table.
- Specifies the index to use when locating records.
- Defines the type of record lock to apply when the records are read.

Syntax

```
{ record [ field-list ] }
[ constant ]
[ [ LEFT ] OUTER-JOIN ]
[ OF table ]
[ WHERE expression ]
[ USE-INDEX index ]
[ USING [ FRAME frame ] field
    [ AND [ FRAME frame ] field ] ... ]
[ SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK ]
[ NO-PREFETCH ]
```

Note: You can specify the OUTER-JOIN, OF, WHERE, USE-INDEX, and USING options in any order. You cannot use *field-list* in an OPEN QUERY statement. You cannot use OUTER-JOIN or EXCLUSIVE-LOCK in a CAN-FIND function.

record

The name of a table or buffer that you named in a DEFINE BUFFER statement.

To access a record in a table defined for multiple databases, you must qualify the record's table name with the database name. Use this syntax to refer to a record in a table for a specific database:

```
dbname.tablename
```

You do not have to qualify the reference if *record* is the name of a defined buffer.

field-list

Specifies a list of fields to include or exclude when you retrieve records using a FOR, DO PRESELECT, or REPEAT PRESELECT statement. Field lists are also available for queries using the DEFINE QUERY statement. Following is the syntax for *field-list*:

```
{    FIELDS [ ( [ field ... ] ) ]
   | EXCEPT [ ( [ field ... ] ) ]
}
```

The FIELDS option specifies the fields you want to include in a record retrieval, and the EXCEPT option specifies the fields that you want to exclude from a record retrieval. The *field* parameter is the name of a single field in the specified table. If *field* is an array reference, the whole array is retrieved even if only one element is specified. Specifying FIELDS with no *field* references causes Progress to retrieve sufficient information to extract the ROWID value for a specified record (returnable using the ROWID function). Specifying EXCEPT with no *field* references or specifying *record* without a *field-list* causes Progress to retrieve a complete record.

This statement retrieves only the name and balance fields of the customer table:

```
FOR EACH customer FIELDS (name balance): DISPLAY name balance.
```

This statement retrieves all fields of the customer table except the name and balance fields:

```
FOR EACH customer EXCEPT (name balance):
    DISPLAY customer EXCEPT name balance.
```

When you specify a field list, Progress might retrieve additional fields or the complete record depending on the type of retrieval operation and the DataServer that provides the record. Thus, Progress:

- Retrieves any additional fields required by the client to complete the record selection.
- Retrieves a complete record when the record is fetched with EXCLUSIVE-LOCK. This ensures proper operation of updates and the local before-image (BI) file. For information on the local BI file, see *OpenEdge Data Management: Database Administration*.
- Retrieves a complete record for DataServers that do not support SHARE-LOCK. For more information, see the appropriate DataServer guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

Note: Always specify fields that you plan to reference in the field list. Only those extra fields that the client requires for record selection are added to the specified field list. Progress distributes record selection between the client and server depending on a number of factors that change with each Progress release. Therefore, never rely on fields that you did not specify but which Progress fetches for its own needs; they might not always be available. There is no additional cost to specify a field in the list that you otherwise expect Progress to provide.

This statement retrieves the customer.cust-num field in addition to those specified in the field lists because it is required to satisfy the inner join between the customer and order tables:

```
FOR EACH customer FIELDS (name),
  EACH order FIELDS (order-num sales-rep) OF customer:
    DISPLAY customer.name customer.cust-num
            order.order-num order.sales-rep.
```

However, do not rely on Progress to always provide such extra fields. For reliability, add the cust-num field to the customer field list as follows:

```
FOR EACH customer FIELDS (name cust-num),
  EACH order FIELDS (order-num sales-rep) OF customer:
    DISPLAY customer.name customer.cust-num
            order.order-num order.sales-rep.
```

constant

The value of a single component, unique, primary index for the record you want. This option is not supported for the OPEN QUERY statement:

```
FIND customer 1.
```

Progress converts this FIND statement with the *constant* option of 1 to the following statement:

```
FIND customer WHERE cust-num = 1.
```

The cust-num field is the only component of the primary index of the customer table.

If you use the *constant* option, you can use it only once in a single Record phrase, and it must precede any other options in the Record phrase.

[LEFT **]** OUTER-JOIN

Specifies a left outer join between *record* and the table (or join) specified by the previous Record phrase(s) of an OPEN QUERY statement. A left outer join combines and returns data from the specified tables in two ways. First, the records selected for the table (or join) on the left side combine with each record selected using the OF or WHERE options from the table on the right (*record*). Second, the records selected for the table (or join) on the left side combine with the Unknown value (?) for the fields from the table on the right (*record*) for which no records are selected using the OF or WHERE options. The join is ordered according to the given sort criteria starting with the left-most table in the query.

Note: If you specify the OUTER-JOIN option, you must also specify the OUTER-JOIN option in all **succeeding** Record phrases of the query to obtain a left outer join. That is, for multiple Record phrases, all joins in the query following your first left outer join must also be left outer joins. Otherwise, the result is an inner join for all records up to the last inner join in the query. For more information, see *OpenEdge Development: Progress 4GL Handbook*.

The OUTER-JOIN option is supported only in the OPEN QUERY statement and in Record phrases specified after the first Record phrase in the OPEN QUERY statement. The LEFT keyword is optional with OUTER-JOIN. If you specify OUTER-JOIN, you must also specify the OF option, WHERE option, or any combination of the OF and WHERE options. These options are required to select *record* (the right-most table) for the specified left outer join. For example:

```
OPEN QUERY q1 PRESELECT EACH customer,
  FIRST order OUTER-JOIN OF customer WHERE order.order-num < 50
  FIRST order-line OUTER-JOIN OF order WHERE order-line.item-num < 15.
```

This query specifies a left outer join between customer and order, and also between that join and order-line. Thus, for each customer record that has no orders or has no orders with an order-num less than 50, the query returns the customer fields and ? for all fields of the order and order-line tables. In addition, if there are no order-line records with item-num less than 15 for any selected customer and order, the query returns ? for all fields of order-line. Otherwise, it returns each customer record along with its first selected order record and order-line record.

In all statements where multiple Record phrases are allowed (including DO, FOR, OPEN QUERY, and REPEAT statements), the default join (without the OUTER-JOIN option) is an inner join between *record* and the table (or join) specified by the previous Record phrase(s). An inner join returns the records selected for the table (or join) on the left side combined with each selected record from the table on the right (*record*). For an inner join, no records are returned for the table (or join) on the left for which no record is selected from the table on the right (*record*).

The following query specifies an inner join between customer and order, and also between that join and order-line. Thus, this query only returns customer records that have at least one order with order-num less than 50 that also have at least one order-line with item-num less than 15, and it returns just the first such order and order-line for each customer record.

```
OPEN QUERY q1 PRESELECT EACH customer,
  FIRST order OUTER-JOIN OF customer WHERE order.order-num < 50
  FIRST order-line OF order WHERE order-line.item-num < 15.
```

Note: If you specify a Record phrase as an inner join, the current Record phrase and all **preceding** Record phrases in the query participate in contiguous inner joins, even if prior Record phrases specify the OUTER-JOIN option. Thus, for multiple Record phrases, all joins in the query up to the right-most inner join result in contiguous inner joins. For more information, see *OpenEdge Development: Progress 4GL Handbook*.

For more information on joins in the 4GL, see *OpenEdge Development: Progress 4GL Handbook*.

OF *table*

Relates *record* to one other table specified by a table or buffer name (*table*). The relationship is based on common field names between *record* and *table* that also participate in a UNIQUE index for either *record* or *table*. When you use OF and the UNIQUE index is multi-field, all fields in the index participate in the match criteria. A reference to *table* must appear in a prior joined Record phrase in the same statement, or remain in scope from a prior record reading statement, such as a FIND statement.

Note: For the OF keyword to properly detect a relationship between two tables, only one such relationship is allowed.

In this example, the OF option relates the order table to the customer table; thus Progress selects the customer record related to the order record currently in use. Progress converts the FIND statement with the OF option to a FIND statement with the WHERE option.

```
PROMPT-FOR order.order-num.
FIND order USING order-num.
DISPLAY order.
FIND customer OF order.
DISPLAY customer.
```

You can use WHERE to access related tables, whether or not the field names of the field or fields that relate the tables have the same name. For example:

```
FIND customer WHERE customer.cust-num =
  order.cust-num.
```

WHERE *expression*

Qualifies the records you want to access. The *expression* is a constant, field name, variable name, or expression whose value you want to use to select records. You can use the WHERE keyword even if you do not supply an *expression*. For example:

```
FOR EACH customer WHERE {*}
```

The WHERE clause may not work the same way against a DataServer as it does against the OpenEdge database. Refer to the appropriate DataServer Guide, *OpenEdge Data Management: DataServer for ODBC* or *OpenEdge Data Management: DataServer for ORACLE*, for additional information on how this feature will perform.

Note: You cannot reference a BLOB or CLOB field in a WHERE clause.

In an OPEN QUERY statement or FOR statement, the WHERE clause can use the CONTAINS operator to reference a field with a word index. This is the syntax for the CONTAINS operator:

```
field CONTAINS search-expression
```

In this syntax, *field* represents a field in which a word index has been defined. The *search-expression* specifies one or more words to search for. It must evaluate to a string with this syntax:

```
"word [ [ & | | | ! | ^ ] word ] ..."
```

Each *word* is a word to search for. The ampersand (&) represents a logical AND; the vertical line (|), exclamation point (!), or caret (^) represent a logical OR.

Here is an example using the CONTAINS clause:

```
FOR EACH item WHERE cat-description CONTAINS "ski":
  DISPLAY item-name cat-description VIEW-AS EDITOR SIZE 60 BY 15.
END.
```

Note: The CONTAINS option is not allowed in a FIND statement. If the session is started with the Version 6 Query (-v6q) parameter, the CONTAINS option is also not allowed in a FOR statement.

Note: For information about compiling, storing, and applying the UTF-8 word-break rules to a database, see *OpenEdge Development: Internationalizing Applications*.

USE-INDEX *index*

Identifies the index you want to use while selecting records. If you do not use this option, Progress selects an index to use based on the criteria specified with the WHERE, USING, OF, or *constant* options.

```
USING [ FRAME frame ] field [ AND [ FRAME frame ] field ] . . .
```

One or more names of fields for selecting records. You must have previously entered each field you name in this option, usually with a PROMPT-FOR statement. The field must be viewed as a fill-in or text widget.

The USING option translates into an equivalent WHERE option:

```
PROMPT-FOR customer.cust-num.
FIND customer USING cust-num.
```

This FIND statement is the same as this statement:

```
FIND customer WHERE customer.cust-num =
  INPUT customer.cust-num.
```

The cust-num field is a non-abbreviated index. However, if the name field is an abbreviated index of the customer table, Progress converts the FIND statement with the USING option. For example:

```
PROMPT-FOR customer.name.
FIND customer USING name.
```

The following statement is a result of the previous one:

```
FIND customer WHERE customer.name
  BEGINS INPUT name.
```

SHARE-LOCK

Tells Progress to put a SHARE-LOCK on records as they are read. Another user can read a record that is share locked, but cannot update it. By default, Progress puts a SHARE-LOCK on a record when it is read (unless it uses a CAN-FIND function), and automatically puts an EXCLUSIVE-LOCK on a record when it is modified (unless the record is already EXCLUSIVE-LOCKed).

In a CAN-FIND function, NO-LOCK is the default. Also, CAN-FIND cannot use EXCLUSIVE-LOCK.

If you use the SHARE-LOCK option and Progress tries to read a record that is EXCLUSIVE-LOCKed by another user, Progress waits to read the record until the EXCLUSIVE-LOCK is released. Progress displays a message to the user of that procedure, identifying the table that is in use, the user ID of the user, and the tty of the terminal using the table.

If you are using a record from a work table, Progress disregards the SHARE-LOCK option.

EXCLUSIVE-LOCK

Tells Progress to put an EXCLUSIVE-LOCK on records as they are read. Other users cannot read or update a record that is EXCLUSIVE-LOCKed, except by using the NO-LOCK option. They can access that record only when the EXCLUSIVE-LOCK is released. Progress automatically puts a SHARE-LOCK on a record when it is read and automatically puts an EXCLUSIVE-LOCK on a record when it is updated.

If a record is read specifying EXCLUSIVE-LOCK, or if a lock is automatically changed to EXCLUSIVE-LOCK by an update, user's read or update will wait if any other user SHARE-LOCKed or EXCLUSIVE-LOCKed the record.

When a procedure tries to use a record that is EXCLUSIVE-LOCKed by another user, Progress displays a message identifying the table that is in use, the user ID of the user, and the tty of the terminal using the table.

If you are using a record from a work table, Progress disregards the EXCLUSIVE-LOCK option. Also, CAN-FIND cannot use the EXCLUSIVE-LOCK option.

Specifying EXCLUSIVE-LOCK causes Progress to retrieve complete records, even when the record is specified with *field-list*.

NO-LOCK

Tells Progress to put no locks on records as they are read, and to read a record even if another user has it EXCLUSIVE-LOCKed.

Another user can read and update a record that is not locked. By default, Progress puts a SHARE-LOCK on a record when it is read (unless it uses a CAN-FIND function, which defaults to NO-LOCK), and automatically puts an EXCLUSIVE-LOCK on a record when it is updated (unless the record is already EXCLUSIVE-LOCKed). A record that has been read NO-LOCK must be reread before it can be updated.

```
DEFINE VARIABLE rid AS ROWID.
rid = ROWID(customer).
FIND customer WHERE
  ROWID(customer) = rid EXCLUSIVE-LOCK.
```

If a procedure finds a record and it places it in a buffer using NO-LOCK and you then refind that record using NO-LOCK, Progress does not reread the record. Instead, it uses the copy of the record that is already stored in the buffer.

When you read records with NO-LOCK, you have no guarantee of the overall consistency of those records because another user might be in the process of changing them. When values are assigned to indexed fields for a newly created record or are modified in an existing record, the index is immediately updated to reflect the change. However the copy of the data record in the buffers used by the database server might not be updated until later in the transaction. For example, the following procedure might display a cust-num of 0 if another user's active transaction has created a record and assigned a value to the indexed field cust-num that is greater than 100:

```
FOR EACH customer WHERE cust-num > 100 NO-LOCK:
  DISPLAY cust-num.
END.
```

If you are using a record from a work table, Progress disregards the NO-LOCK option.

NO-PREFETCH

Specifies that only one record is sent across the network at a time. If you specify *field-list*, only the specified fields and any additional fields required for record selection are sent. If you do not specify this option, Progress can send more than one record from the server to the client in each network packet.

Examples In the r-recph.p procedure, there are two Record phrases that make an inner join between the customer and order tables.

r-recph.p

```
FOR EACH customer FIELDS (cust-num name credit-limit)
        WHERE credit-limit GE 50000,
    EACH order FIELDS (order-num order-date terms) OF customer:
    DISPLAY customer.cust-num customer.name credit-limit
      order.order-num order-date order.terms.
END.
```

Using these Record phrases, the FOR EACH block reads a customer record only if it has a credit-limit value greater than 50000 and at least one order record associated with it.

r-recph2.p

```
REPEAT:
    FIND NEXT customer USE-INDEX country-post WHERE name BEGINS "S"
        EXCLUSIVE-LOCK.
    UPDATE name country postal-code phone.
END.
```

In the r-recph2.p procedure, there is one Record phrase:

```
customer USE-INDEX country-post WHERE name BEGINS "S" EXCLUSIVE-LOCK"
```

Using the zip index named country-post rather than the cust-num index (the primary index for the customer table), the FIND statement reads only those customer records that have a name that begins with an s. The FIND also places an EXCLUSIVE-LOCK on each record as it is read. This lock is released at the end of the REPEAT block.

In the output of this procedure, all the customer names begin with s and the customers are displayed in order by country and then postal code.

Notes

- Specifying a field list (*field-list*) for *record* can increase the performance of remote (network) record retrieval substantially over specifying *record* alone. For more information, see *OpenEdge Development: Progress 4GL Handbook*.
- If you reference an unfetched database field at run time, Progress raises the ERROR condition. Progress does not perform a compile-time check to ensure that the field is fetched because the compiler cannot reliably determine how a particular record will be read (that is, whether it is retrieved using a FIND statement, retrieved with or without a field list, including additional fields to satisfy join conditions, etc.).
- Do not use a field list if you delete or update the record shortly after the record retrieval. Otherwise, Progress reads the whole record, again, to complete the delete or update.
- You can specify the Field List Disable (`-fldisable`) startup parameter to cancel field list retrieval and force Progress to retrieve complete records. This is a runtime client session parameter that is especially useful for deployed applications whose database triggers are later redefined to reference unfetched fields (raising the ERROR condition). Using -fldisable provides a workaround that allows the application to run (although more slowly) until the application can be fixed.
- You cannot specify field lists or joins in a FIND statement, or specify field lists in an OPEN QUERY statement.
- You cannot use the CONTAINS operator with a temporary table.
- If used, the CONTAINS operator must appear in the outer-most WHERE expression. You can combine it with other expressions at the outer level using the AND and OR operators. However, you cannot apply the NOT operator to a CONTAINS expression.
- You cannot reference a BLOB or CLOB field in a WHERE clause.
- Temporary tables and work tables can be used in join conditions specified with the OF option as long as the OF option requirements identified earlier in this section have been satisfied.

- Do not compare case-sensitive data with case-insensitive data in a WHERE expression. Progress both cannot determine the results and does not raise the ERROR condition if you specify data with mixed case sensitivity in selection criteria because:
  - Mixed case sensitivity in selection criteria is handled differently by different DataServers.
  - Mixed case-sensitivity results for the same DataServer can be different depending on whether the query is resolved on the client or the server.
  - Some national languages do not support the concept of case sensitivity.

  Thus, such queries cannot be reliably resolved in any way.
- SpeedScript – The only invalid option is USING FRAME.

See also DEFINE QUERY statement, DO statement, FIND statement, FOR statement, OPEN QUERY statement, REPEAT statement

RECORD-LENGTH function

Returns the length of a record in a buffer.

Syntax

```
RECORD-LENGTH ( buffer )
```

buffer

A database buffer containing a record.

Note

The RECORD-LENGTH function is especially useful when implementing 4GL-based database replication, which involves storing entire database records in log record fields. Progress limits records to 32K. Before you transfer a record to a raw field in another record, you can use RECORD-LENGTH to ensure that you are not expanding the record beyond the 32K limit.

See also

RAW-TRANSFER statement

REJECTED function

Returns the current REJECTED attribute setting for a ProDataSet temp-table buffer.

Syntax

```
REJECTED( buffer-name )
```

buffer-name

The name of a ProDataSet temp-table buffer.

Notes

- This function is typically used with the SAVE-ROW-CHANGES( ) method.
- The REJECTED function corresponds to the REJECTED attribute.
- You can invoke the REJECTED function from within a WHERE clause (unlike the corresponding attribute).

RELEASE statement

Verifies that a record complies with mandatory field and unique index definitions. It clears the record from the buffer and unites it to the database if it has been changed.

Syntax

```
RELEASE record [ NO-ERROR ]
```

record

The name of a record buffer.

To use RELEASE with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

NO-ERROR

Specifies that any errors that occur in the attempt to release the record are suppressed. After the RELEASE statement completes, you can check the ERROR-STATUS system handle for information on any errors that occurred.

Example

The following example uses a browse widget to scan customer records. Records within the browse are read with NO-LOCK. If you choose the Update Customer button, the CHOOSE trigger starts a transaction and applies an EXCLUSIVE-LOCK to the customer record. When you have completed any updates, the procedure displays the new values in the browse widget and then executes a RELEASE statement. This ensures that the lock is released when the transaction ends.

r-rels.p

```
DEFINE BUTTON upd-cust LABEL "Update Customer".
DEFINE BUTTON exit-app LABEL "Exit".
DEFINE VARIABLE methRtn AS LOGICAL.
DEFINE VARIABLE curr-cust AS ROWID.
DEFINE QUERY seq-cust FOR customer.
DEFINE BROWSE brow-cust QUERY seq-cust DISPLAY Cust-num Name WITH 10 DOWN.

FORM
   upd-cust exit-app SKIP(1)
   brow-cust
   WITH FRAME main-frame.

FORM
   customer EXCEPT comments
   WITH FRAME curr-frame COLUMN 40.

OPEN QUERY seq-cust FOR EACH customer.

ON VALUE-CHANGED OF brow-cust
   DO:
      DISPLAY customer EXCEPT comments WITH FRAME curr-frame SIDE-LABELS.
      curr-cust = ROWID(customer).
   END.

ON CHOOSE OF upd-cust
   DO: /* TRANSACTION */
      FIND customer WHERE ROWID(customer) = curr-cust EXCLUSIVE-LOCK.
      UPDATE customer WITH FRAME cust-frame VIEW-AS DIALOG-BOX
         TITLE "Customer Update".
      methRtn = brow-cust:REFRESH().
      DISPLAY customer EXCEPT comments WITH FRAME curr-frame SIDE-LABELS.
      RELEASE customer.
   END.

ENABLE ALL WITH FRAME main-frame.
APPLY "VALUE-CHANGED" TO brow-cust.
PAUSE 0 BEFORE-HIDE.
WAIT-FOR CHOOSE OF exit-app OR WINDOW-CLOSE OF DEFAULT-WINDOW.
```

If you omit the RELEASE statement in this example, the EXCLUSIVE-LOCK is downgraded to a SHARE-LOCK at the end of the transaction. This prevents other uses from updating that record. The SHARE-LOCK is released when you change the iteration of the browse.

Notes

- An ERROR occurs if the validation of the record fails. This can happen only with newly created records.
- If a record has been modified, the RELEASE statement causes a WRITE event and fires any related WRITE trigger to execute. All WRITE triggers execute before the record is actually written. If a WRITE trigger fails (or executes a RETURN statement with the ERROR option), the corresponding record is not written or released and the ERROR condition is raised for the RELEASE statement. See *OpenEdge Development: Progress 4GL Handbook* for more information on database triggers.
- See *OpenEdge Development: Progress 4GL Handbook* for more information on transactions.

RELEASE EXTERNAL statement

Frees (that is, unloads from memory) a dynamic link library (DLL) or UNIX shared library.

Syntax

```
RELEASE EXTERNAL [ PROCEDURE ] "dll-name"
```

[PROCEDURE]

An optional “noise” keyword that does not affect the statement’s behavior in any way.

dll-name

A character string representing the name of the DLL or UNIX shared library.

Example

To free the dll `mystuff.dll`, code the following:

```
RELEASE EXTERNAL PROCEDURE "mystuff.dll".
```

RELEASE OBJECT statement

Releases the specified COM object (Automation object or ActiveX control) and removes all internal structures associated with the handle to the object.

Syntax

```
RELEASE OBJECT COM-hdl-var [ NO-ERROR ]
```

COM-hdl-var

A COM-HANDLE variable that references a valid COM object.

NO-ERROR

Specifies that any errors that occur in the attempt to release the object are suppressed. After the RELEASE OBJECT statement completes, you can check the ERROR-STATUS system handle for information on any errors that occurred.

Example

This procedure fragment shows a control named hc_CmdButton being loaded into a control-frame and the handle to the control (controlHdl) being obtained using the control name (hc_CmdButton) property. Later, it releases the control and deletes the parent control-frame widget (CFWidHdl).

```
DEFINE VARIABLE CFWidHdl AS WIDGET-HANDLE.
DEFINE VARIABLE CFComHdl AS COM-HANDLE.
DEFINE VARIABLE controlHdl AS COM-HANDLE.

/* Create frame foo ... */

CREATE CONTROL-FRAME CFWidHdl
  ASSIGN
    FRAME = FRAME foo:HANDLE
    NAME = "ctlFrame1".
CFComHdl = CFWidHdl:COM-HANDLE.
CFComHdl:LoadControls(hc_CmdButton.wrx, "hc_CmdButton").

controlHdl = CFComHdl:hc_CmdButton.
controlHdl:BgColor = RGB-VALUE(0,128,0).

/* do some more stuff ... WAIT-FOR ... */
RELEASE OBJECT controlHdl. /* NOTE: Not really necessary */
DELETE WIDGET CFWidHdl.
```

For an example of the RELEASE OBJECT statement applied to Automation objects, see the CREATE automation object statement entry.

Notes

- After this statement completes, any other component handles that reference the object are invalid. If you attempt to reference the object using one of these handles, Progress returns an invalid handle error. It is also possible for a newly instantiated COM object to get the same handle as one that has been released. Progress does not detect that this occurs. In this case, the “old” handle is valid, but it references a different control. Thus, it is a good practice to set any COM-HANDLE variables that reference a released COM object to the Unknown value (?).
- The released COM object remains active as long as any other COM object has a valid reference to it. In the case of an ActiveX control, the parent control-frame is a COM object that references the control. All other component handle references you establish in the OpenEdge session represent a second reference to the COM object. Thus, when you release one of these component handles, the released COM object remains active as long as the parent control-frame COM object is still active. To release the parent control-frame COM object and complete the release of the ActiveX control, you must follow any release of the ActiveX control by a delete of the parent control-frame widget.
- When you delete a control-frame widget, Progress releases all associated ActiveX controls automatically, whether or not you release them individually.
- When the session ends, Progress automatically releases any active COM objects you have not released individually.

See also

CREATE automation object statement, DELETE WIDGET statement, DELETE WIDGET-POOL statement

REPEAT statement

Begins a block of statements that are processed repeatedly until the block ends in one of several ways.

Block properties

Iteration, record scoping, frame scoping, transactions by default.

Syntax

```
[ label : ] REPEAT
  [ FOR record [ , record ] ... ]
  [ preselect-phrase ]
  [ query-tuning-phrase ]
  [ variable = expression1 TO expression2 [ BY k ] ]
  [ WHILE expression ]
  [ TRANSACTION ]
  [ on-endkey-phrase ]
  [ on-error-phrase ]
  [ on-quit-phrase ]
  [ on-stop-phrase ]
  [ frame-phrase ]
```

`FOR record [ , record ] . . .`

Names a record buffer and scopes the buffer to the block. The scope of a record determines when the buffer is cleared and the record is written back to the database. See *OpenEdge Development: Progress 4GL Handbook* for more information on record scoping and blocks.

To access a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

preselect-phrase

Goes through a table to select the records that meet the criteria you specify in a *record-phrase*. PRESELECT creates a temporary index that contains pointers to each of the preselected records in the database table. You can then use other statements, such as FIND NEXT, to process those records. Following is the syntax for *preselect-phrase*:

```
PRESELECT
  [ EACH | FIRST | LAST ] record-phrase
    [ , { EACH | FIRST | LAST } record-phrase ] ...
  [ [ BREAK ] { BY expression [ DESCENDING ] } ... ]
```

For more information, see the PRESELECT phrase reference entry.

query-tuning-phrase

Allows programmatic control over the execution of a DataServer query. Following is the syntax for the *query-tuning-phrase*:

```
QUERY-TUNING
  (
    [ BIND-WHERE | NO-BIND-WHERE ]
    [ CACHE-SIZE integer ]
    [ DEBUG { SQL | EXTENDED } | NO-DEBUG ]
    [ INDEX-HINT | NO-INDEX-HINT ]
    [ JOIN-BY-SQLDB | NO-JOIN-BY-SQLDB ]
    [ LOOKAHEAD | NO-LOOKAHEAD ]
    [ SEPARATE-CONNECTION | NO-SEPARATE-CONNECTION ]
  )
```

For more information, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

variable = *expression1* TO *expression2* [BY *k*]

Indicates the name of a field or variable whose value you are incrementing in a loop. The *expression1* is the starting value for *variable* on the first iteration of the loop. The *k* is the amount to add to *variable* after each iteration and must be a constant. When *variable* exceeds *expression2* (or is less than *expression2* if *k* is negative), the loop ends. Because *expression1* is compared to *expression2* at the start of the first iteration of the block, the block can be executed zero times. The *expression2* is reevaluated with each iteration of the block.

WHILE *expression*

Indicates the condition during which the REPEAT block processes the statements within it. The block iterates as long as the condition specified by the expression is TRUE. The expression is any combination of constants, field names, and variable names that yield a logical value.

TRANSACTION

Identifies the REPEAT block as a system transaction block. Progress starts a system transaction for each iteration of a transaction block if there is no active system transaction. See *OpenEdge Development: Progress 4GL Handbook* for more information on transactions.

on-endkey-phrase

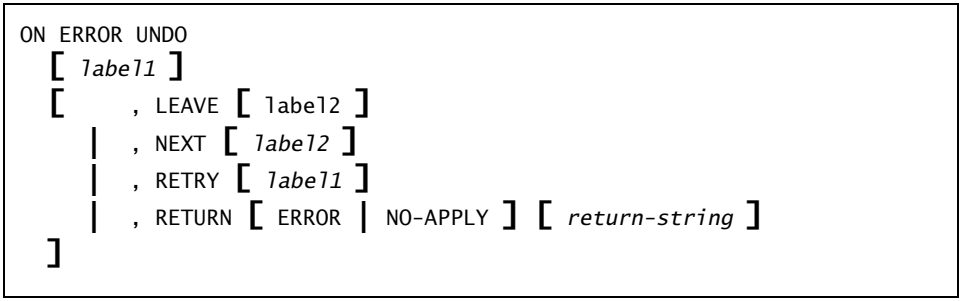
Describes the processing that takes place when the ENDKEY condition occurs during a block. Following is the syntax for the ON ENDKEY phrase:

```
ON ENDKEY UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
       | , NEXT [ label2 ]
       | , RETRY [ label1 ]
       | , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON ENDKEY phrase reference entry.

on-error-phrase

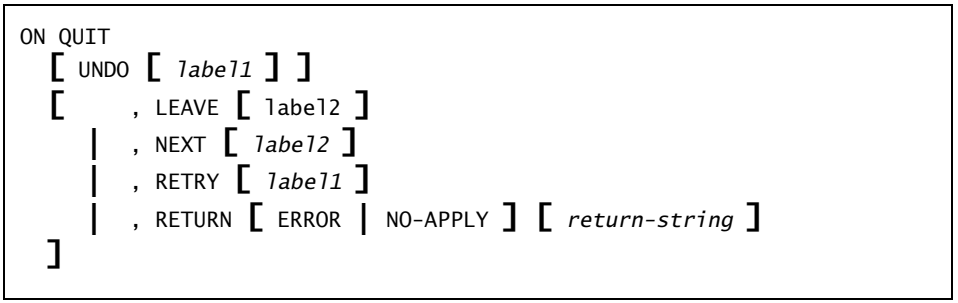
Describes the processing that takes place when there is an error during a block. This is the syntax for the ON ERROR phrase:



For more information, see the ON ERROR phrase reference entry.

on-quit-phrase

Describes the processing that takes place when a QUIT statement is executed during a block. This is the syntax for the ON QUIT phrase:



For more information, see the ON QUIT phrase reference entry.

on-stop-phrase

Describes the processing that takes place when the STOP conditions occurs during a block. This is the syntax for the ON STOP phrase:

```
ON STOP UNDO
  [ label1 ]
  [      , LEAVE [ label2 ]
     |   , NEXT [ label2 ]
     |   , RETRY [ label1 ]
     |   , RETURN [ ERROR | NO-APPLY ] [ return-string ]
  ]
```

For more information, see the ON STOP phrase reference entry.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information, see the Frame phrase reference entry.

Example

In this menu procedure, if you press **END-ERROR** or **ENDKEY** when the procedure prompts you for your menu selection, any data you have entered as a selection is undone and the procedure continues to prompt you for a menu selection:

r-rpt.p

```
DEFINE VAR Selection AS INTEGER FORMAT "9".

FORM skip(3)
      "0 - Exit" at 32
      "1 - Edit Customer File" at 32
      "2 - List Customer File" at 32
      "3 - Edit Item File" at 32
      "4 - List Item File" at 32
      "Enter Choice" TO 30 Selection AUTO-RETURN
      HEADER "Application Name"  "Master Menu" AT 34  "Company" TO 79
      WITH NO-BOX NO-LABELS CENTERED FRAME menu.

/* Create the procedures that are called from the following block. */

REPEAT ON ENDKEY UNDO, RETRY:
    UPDATE Selection WITH FRAME menu.
    HIDE FRAME menu.

    CASE(Selection):
      WHEN 0 THEN
        LEAVE.
      WHEN 1 THEN
        RUN custedit.p.
      WHEN 2 THEN
        RUN custrpt.p.
      WHEN 3 THEN
        RUN itemedit.p.
      WHEN 4 THEN
        RUN itemrpt.p.
      OTHERWISE DO:
        BELL.
        MESSAGE "Not a valid choice. Try again.".
      END.
    END CASE.

END.  /* REPEAT  */
```

Notes

- Within a REPEAT block, if you are using the FIND NEXT or FIND PREV statement and you change the value of an index field, Progress makes that change in the index table at the end of the UPDATE or SET statement. Therefore, if you change the value so that the record appears later in the index table, you will see the record again if you FIND NEXT. If you change the value so that the record appears earlier in the index table, you see the record again if you FIND PREV.

```
REPEAT:
  FIND NEXT customer.
  UPDATE cust-num.
END.
```

In this example, if you change customer 1 to customer 300, you see that customer record again at the end of the procedure.

When you use the PRESELECT option, Progress builds a special index table that is not updated when index values change. For example, add the PRESELECT option to the above example:

```
REPEAT PRESELECT EACH customer:
  FIND NEXT customer.
  UPDATE cust-num.
END.
```

In this example, if you change customer 2 to customer 200, you do not see that customer record until you look it up with a new procedure.

- SpeedScript – The invalid options are: *on-endkey-phrase* and *on-quit-phrase*.

See also

DO statement, END statement, Frame phrase, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, ON STOP phrase

REPLACE function

Returns a string with specified substring replacements.

Syntax

```
REPLACE ( source-string , from-string , to-string )
```

source-string

Specifies the base string to make replacements in. The *source-string* parameter can be any expression that evaluates to a string or a LONGCHAR. The REPLACE function does not change the value of *source-string* itself; the function returns the string with replacements.

from-string

Specifies the substring to replace. The *from-string* parameter can be any expression that evaluates to a string or a LONGCHAR. Each occurrence of *from-string* within *source-string* is replaced.

to-string

Specifies the replacement substring. The *to-string* parameter can be any expression that evaluates to a string or a LONGCHAR. Each occurrence of *from-string* in *source-string* is replaced by *to-string*.

Example

The following example uses the REPLACE function to replace the string “user” with an actual user ID, if available:

r-repl.p

```
DEFINE VARIABLE greeting AS CHARACTER FORMAT "x(40)"
      INITIAL "Starting user's session . . . ".
IF USERID("DICTDB") < > ""
THEN greeting = REPLACE(greeting, "user", USERID("DICTDB")).

DISPLAY greeting WITH NO-LABELS.
```

Notes

- The REPLACE function replaces all occurrences of *from-string* within *source-string*. After replacing a substring, the REPLACE function resumes searching the string after the inserted text. Thus, the inserted text is not recursively searched (in whole or in part) for *from-string*.
- The search for occurrences of *from-string* within *source-string* is not case sensitive, unless one of the three values used in the function (*source-string*, *to-string*, or *from-string*) is a case-sensitive field or variable.

See also

OVERLAY statement, SUBSTITUTE function, SUBSTRING function

REPOSITION statement

Repositions the cursor associated with a specific query. The query must be associated with a browse widget or defined with the SCROLLING option. The next record to be retrieved is the record following the cursor position.

Syntax

```
REPOSITION query
  {     TO ROWID rowid1 [ , rowid2 ] ... [ NO-ERROR ]
      | TO RECID recid [ NO-ERROR ]
      | ROW n
      | FORWARDS n
      | BACKWARDS n
  }
```

query

The name of the query to reposition. The query must be open.

TO ROWID *rowid1* **[** , *rowid2* **]** **...** **[** NO-ERROR **]**

Repositions the query to the join levels that correspond to the rowids you specify. *rowid1* represents the rowid of the top level of join, *rowid2* represents the rowid of the next level of join, etc. You can specify any number of rowids up to the number of join levels. If you specify fewer rowids than the number of join levels, Progress still repositions the query to the join levels that correspond to the rowids you specify, but arranges the remaining join levels arbitrarily.

NO-ERROR suppresses any error messages that result from specifying an illegal value or a value that does not identify any records returned by the query. To test whether an error occurred during a reposition operation, use the ERROR-STATUS handle.

`TO RECID recid [ NO-ERROR ]`

Similar to the TO ROWID option, except that the value *recid* is an expression that evaluates to a RECID value, and you can specify only one recid. Supported only for backward compatibility.

NO-ERROR suppresses any error messages that result from specifying an illegal value or a value that does not identify any records returned by the query. To test whether an error occurred during a reposition operation, use the ERROR-STATUS handle.

`TO ROW n`

Repositions the cursor to before the specified row in the result list of the query. The value *n* must be an INTEGER expression that identifies a row in the result list. You cannot use this option with a query opened with the INDEXED-REPOSITION option.

`FORWARDS n`

Moves the cursor from its current position in the result list to a new position *n* records forward, where *n* represents an INTEGER expression.

REPOSITION FORWARDS always places the cursor between two rows. For example:

- If the cursor is on a row—say, row 5—REPOSITION FORWARDS 1 moves the cursor to row 6, then to half way between rows 6 and 7. From this position, GET PREVIOUS moves the cursor to row 6, while GET-NEXT moves the cursor to row 7.
- If the cursor is already between two rows—say, between rows 5 and 6—REPOSITION FORWARDS 1 moves the cursor to half way between rows 6 and 7. From this position, GET PREVIOUS moves the cursor to row 6, while GET-NEXT moves the cursor to row 7.

`BACKWARDS` *n*

Moves the cursor from its current position in the result list to a new position *n* records back, where *n* represents an INTEGER expression.

REPOSITION BACKWARDS always places the cursor between two rows. For example:

- If the cursor is on a row— say, row 5 —REPOSITION BACKWARDS 1 moves the cursor to row 4, then to half way between rows 4 and 5. From this position, GET PREVIOUS moves the cursor to row 4, while GET-NEXT moves the cursor to row 5.
- If the cursor is already between two rows— say, between rows 5 and 6 — REPOSITION BACKWARDS 1 moves the cursor to half way between rows 4 and 5. From this position, GET PREVIOUS moves the cursor to row 4, while GET-NEXT moves the cursor to row 5.

Example The following example uses the REPOSITION statement to move forward or backward within a query:

r-repos.p

```
DEFINE QUERY  q-order FOR customer, order SCROLLING.
DEFINE BUTTON b_quit     LABEL "Quit".
DEFINE BUTTON b_frwd     LABEL "FORWARD".
DEFINE BUTTON b_back     LABEL "BACKWARD".
DEFINE VAR num AS INTEGER INIT 1 NO-UNDO.

FORM b_frwd b_back b_quit
  WITH FRAME butt-frame ROW 1.

ON CHOOSE OF b_back, b_frwd
DO:
  PROMPT-FOR num  LABEL "Records To Skip"
    WITH FRAME pos-info CENTERED ROW 5 overlay.
  HIDE FRAME pos-info NO-PAUSE.
  IF SELF:LABEL = "BACKWARD" THEN
    REPOSITION q-order BACKWARDS INPUT num + 1.
  ELSE REPOSITION q-order FORWARDS INPUT num - 1.
  RUN getone.
END.

OPEN QUERY q-order FOR EACH customer,
  EACH order OF customer NO-LOCK.

RUN getone.

ENABLE b_back b_frwd b_quit WITH FRAME butt-frame.
WAIT-FOR CHOOSE OF b_quit OR WINDOW-CLOSE OF CURRENT-WINDOW.

PROCEDURE getone:
  GET NEXT q-order.
  IF NOT AVAILABLE(customer) THEN
  DO:
    REPOSITION q-order BACKWARDS 1.
    GET NEXT q-order.
  END.
  DISPLAY customer.cust-num customer.name skip
          order.order-num order.order-date
    WITH FRAME order-info CENTERED ROW 5 SIDE-LABELS OVERLAY.
END PROCEDURE.
```

Notes

- The REPOSITION statement does not fetch a record, except when the query is associated with a browse. The REPOSITION statement positions the cursor for the query so that a subsequent GET NEXT statement fetches the specified record, and GET PREV fetches the record before it.
- If you reposition a query associated with a browse widget, the browse widget data is refreshed with the record after the new position at the top.
- If you try to position the cursor outside the list of records that satisfy the query, Progress does **not** raise the ERROR condition. If you try to position the cursor before the first record, Progress positions the query to just before the first record. If you try to position the cursor beyond the last record, Progress positions it just beyond the last record.
- The REPOSITION statement might be slow if the record you position to has not yet been fetched.
- The REPOSITION TO ROWID statement might be especially slow. If the record has not yet been fetched, Progress performs a series of GET NEXT operations until the record is found. You can optimize the performance of a REPOSITION TO ROWID statement by opening the query using the INDEXED-REPOSITION option of the OPEN QUERY statement.
- The INDEXED-REPOSITION option of the OPEN QUERY statement, followed by REPOSITION TO ROWID or GET LAST, causes the query results list to change dramatically. Subsequent use of the CURRENT-RESULT-ROW or NUM-RESULTS functions might produce unknown or unexpected results.
- The order of the records in the query is determined by the options specified in the OPEN QUERY statement.
- SpeedScript – The *on-endkey-phrase* and the *on-quit-phrase* do not apply.

See also

CLOSE QUERY statement, CURRENT-RESULT-ROW function, DEFINE QUERY statement, GET statement, NUM-RESULTS function, OPEN QUERY statement

RETRY function

Returns a TRUE value if the current block is being reprocessed after a previous UNDO, RETRY.

Syntax

```
RETRY
```

Example

This procedure bypasses the display of the customer data when the REPEAT block is retried (if user changes the customer data and does not specify a country). When you run this procedure, notice that even though the procedure has undone any data that you entered (if you did not specify a country), the data still appears in the window. The data is saved in the screen buffers, but it is not stored in the customer record buffer. If you do not use the RETRY function, Progress reprocesses the DISPLAY statement and display the previous values for the customer fields, overwriting the data that was entered in error.

r-retry.p

```
REPEAT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING cust-num.

    IF NOT RETRY
      THEN DISPLAY name address city state country.
    ELSE DISPLAY country.

    SET name address city state country.
    IF country = "" THEN UNDO, RETRY.
END.
```

Notes

- Using the RETRY function in a block turns off the default error processing, which result in no infinite loop protection for the block.
- For more information on retry processing, see *OpenEdge Development: Progress 4GL Handbook*.

See also

UNDO statement

RETURN statement

Leaves the local or remote procedure block and returns to the calling procedure. If there is no calling procedure, RETURN returns to the Procedure Editor or other ADE tool that invoked the procedure.

For more information on remote procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

Syntax

```
RETURN
  [ ERROR | NO-APPLY ]
  [ return-value ]
```

ERROR

Causes an ERROR condition in the calling block. This causes the ERROR condition to be raised for the RUN statement in the calling procedure. You can use the ERROR option only in a procedure or a database trigger block. Any values that are set for OUTPUT or INPUT-OUTPUT parameters before the RETURN ERROR executes are not returned to the calling procedure.

NO-APPLY

Suppresses the default behavior for the current user-interface event. For example, the default behavior for a character code key press in a fill-in field is to echo the character in the field. If you execute RETURN NO-APPLY in a trigger, this behavior is not performed. You can use the NO-APPLY option in a user-interface trigger block or within an internal procedure.

return-value

The value that RETURN returns to the calling procedure. RETURN appearing in a user-defined function returns an expression whose type matches the return type of the function. RETURN not appearing in a user-defined function returns a CHARACTER expression. To access *return-value* from the calling procedure, use the RETURN-VALUE function.

Examples

The `r-fact.p` procedure is called recursively because (*n* factorial) is *n* * ((*n* - 1) factorial). The `r-fact.p` procedure first checks that the input value is valid. If the value is invalid, it returns a message to the caller. Note that `r-return.p` checks the RETURN-VALUE immediately after running `r-fact.p`. If a message is returned, `r-return.p` displays that message.

The procedure `r-return.p` accepts an integer as input and then runs `r-fact.p` to calculate the factorial of that integer. The factorial of a number is the result of multiplying together all of the integers less than or equal to that number (for example: 3 factorial is 3 * 2 * 1 = 6). The `r-fact.p` procedure is called recursively because *n* factorial is *n* * (*n* -1) factorial.

r-return.p

```
DEFINE NEW SHARED VARIABLE nfact AS INTEGER LABEL "N Factorial"
                        FORMAT ">,>>>,>>>,>>9".
DEFINE VARIABLE n AS INTEGER FORMAT "->9" LABEL "N".

REPEAT:
    SET n SPACE(5).
    nfact = n.
    RUN r-fact.p.
    IF RETURN-VALUE <> ""
    THEN DO:
       BELL.
       MESSAGE RETURN-VALUE.
    END.
    ELSE DISPLAY nfact.
END.
```

r-fact.p

```
DEFINE SHARED VARIABLE nfact AS INTEGER.
DEFINE VARIABLE i AS INTEGER.

IF nfact < 0
THEN RETURN "The value is negative.".

IF nfact > 12
THEN RETURN "The calculated value won't fit in an integer.".

i = nfact.
nfact = nfact - 1.

IF nfact <= 1 THEN DO:
    nfact = i.
    RETURN.
END.

RUN r-fact.p.

nfact = nfact * i.

RETURN.
```

Note that this is not the most efficient way to calculate factorials, but in other applications, such as bill of material explosions, recursive procedures are very effective.

Notes

- The RETURN-VALUE function provides the value returned by the most recently executed RETURN statement of a local or remote procedure.
- If the procedure executing the RETURN statement is called asynchronously, the client can access the return value and ERROR condition in the associated event procedure. For more information on event procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

See also

CREATE SAX-READER statement, FUNCTION statement, ON ENDKEY phrase, ON ERROR phrase, ON QUIT phrase, ON STOP phrase, RETURN-VALUE function

RETURN-VALUE function

Provides the value returned by the most recently executed RETURN statement of a local or remote procedure.

Syntax

```
RETURN-VALUE
```

Example

For an example of the RETURN-VALUE function, see the RETURN statement reference entry.

Notes

- The returned value has the CHARACTER data type.
- If no value was returned by the most recently executed RETURN statement, RETURN-VALUE returns an empty string ("").
- If you have a procedure which does not end with the RETURN statement, the value in RETURN-VALUE will be the value of the last executed RETURN statement. RETURN-VALUE is not cleared if there is no RETURN statement.
- For more information on remote procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

See also

CREATE SAX-READER statement, RETURN statement

RGB-VALUE function

Returns an integer that represents a combination of a red, green, and blue color value. This function allows you to define an arbitrary color, expanding beyond those colors defined in the color table.

Note: Does not apply to SpeedScript programming.

Syntax

```
RGB-VALUE ( redval , greenval , blueval )
```

redval, *greenval*, *blueval*

Identifies red, green, and blue color values which can be combined to define a unique color value.

Example

The following code fragment shows how to set the background color of an ActiveX control:

```
DEFINE VARIABLE hdlControl AS COM-HANDLE.
/* Complete code to get a handle to a control in a control-frame.*/
.
.
.
hdlControl:BackColor = RGB(128, 0, 256).
```

For detailed information on programming ActiveX Controls, see *OpenEdge Development: Programming Interfaces* manual.

Note

The RGB-VALUE function is generally most useful when it is used with ActiveX Controls.

See also

COLOR-TABLE system handle

RIGHT-TRIM function

Removes trailing white space, or other specified characters, from a CHARACTER or LONGCHAR expression.

Syntax

```
RIGHT-TRIM ( expression [ , trim-chars ] )
```

expression

An expression (a constant, field name, variable name, or expression) whose value is a CHARACTER or LONGCHAR. If *expression* is a case-sensitive variable, Progress performs a case-sensitive trim. If *expression* is a LONGCHAR, the result is in the same code page.

trim-chars

A character expression that specifies the characters to trim from *expression*. If you do not specify *trim-chars*, the RIGHT-TRIM function removes spaces, tabs, line feeds, and carriage returns.

Example The following example shows the effects of the TRIM, RIGHT-TRIM, and LEFT-TRIM functions:

r-ltrim.p

```
DEFINE BUTTON b_left LABEL "Left Trim".
DEFINE BUTTON b_right LABEL "Right Trim".
DEFINE BUTTON b_trim  LABEL "Trim".
DEFINE BUTTON b_quit  LABEL "Quit" AUTO-ENDKEY.

DEFINE VARIABLE i AS INTEGER NO-UNDO.

DEFINE VARIABLE txt AS CHARACTER FORMAT "X(26)" INIT
  "***** This is a test *****".

DEFINE FRAME butt-frame
  txt i LABEL "String Length" SKIP(2)
  b_left b_right b_trim b_quit
WITH CENTERED TITLE "Original Text String".

DEFINE FRAME trimed-frame
  txt LABEL "Trimed Text"
  i   LABEL "Length"
WITH CENTERED.

ON CHOOSE OF b_trim, b_right, b_left IN FRAME butt-frame
DO:
  FRAME trimed-frame:TITLE  = "Data After " + SELF:LABEL.
  DISPLAY TRIM(txt, "* ") WHEN SELF:LABEL = "Trim" @  txt
    LENGTH(TRIM(txt, "* ")) WHEN SELF:LABEL = "Trim" @ i
    LEFT-TRIM(txt,"* ") WHEN SELF:LABEL = "Left Trim"  @ txt
    LENGTH(LEFT-TRIM(txt,"* ")) WHEN SELF:LABEL = "Left Trim" @ i
    RIGHT-TRIM(txt, "* ") WHEN SELF:LABEL = "Right Trim" @ txt
    LENGTH(RIGHT-TRIM(txt, "* ")) WHEN SELF:LABEL = "Right Trim"  @ i
  WITH FRAME trimed-frame.
END.

ENABLE b_left b_right b_trim b_quit WITH FRAME butt-frame.

i = LENGTH(txt).
DISPLAY txt i WITH FRAME butt-frame.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.
```

Notes

- The RIGHT-TRIM function is similar to the TRIM function except that it trims characters only from the right end of the string.
- If *expression* is a case-sensitive field or variable, then *trim-chars* is also treated as case sensitive. Otherwise, *trim-chars* is not case sensitive.
- The RIGHT-TRIM function is double-byte enabled. The specified *expression* and *trim-chars* arguments can contain double-byte characters. RIGHT-TRIM does not remove double-byte space characters by default.

See also LEFT-TRIM function, TRIM function

ROUND function

Rounds a decimal expression to a specified number of places after the decimal point.

Syntax

```
ROUND ( expression , precision )
```

expression

A decimal expression.

precision

A non-negative integer expression whose value is the number of places you want in the decimal result of the ROUND function.

Example

This procedure increases all credit-limit values by 10 percent, rounding those values to the nearest $100:

r-round.p

```
FOR EACH customer:
    DISPLAY cust-num name credit-limit.
    credit-limit = ROUND( (credit-limit * 1.1) / 100 ,0) * 100.
    PAUSE.
    DISPLAY credit-limit.
END.
```

See also

TRUNCATE function

ROW-STATE function

Returns an integer value that represents the current change state of a static ProDataSet temp-table buffer.

Syntax

```
ROW-STATE( buffer-name )
```

buffer-name

The name of a ProDataSet temp-table buffer (preferably a before-image temp-table buffer).

Notes

- The ROW-STATE function corresponds to the ROW-STATE attribute.
- When the TRACKING-CHANGES attribute is set to TRUE for a ProDataSet temp-table, Progress tracks changes to the data in that temp-table using a before-image temp-table that contains the original version of each modified row. You can think of the temp-table itself as the after-image because it contains the latest version of each row.

  Every row in the after-image table that has been modified or created corresponds to a row in the before-image table. Deleted rows do not appear in the after-image table, because it reflects the current state of the data. Every row in the before-image table has a non-zero ROW-STATE, because every row is the before-image of a deleted, created, or modified row in the after-image table. Unchanged rows do not appear in the before-image table.

  You can use the ROW-STATE function on each row in either the after-image table or the before-image table to determine whether a row has changed and how it has changed.

- The possible return values can be expressed as compiler constants. Table 44 lists these values.

Table 44: Row state values

Compiler constant	Value	Description
ROW-UNMODIFIED	0	The row was not modified.
ROW-DELETED	1	The row was deleted.
ROW-MODIFIED	2	The row was modified.
ROW-CREATED	3	The row was created.

- The ROW-STATE function returns the Unknown value (?) when the specified temp-table buffer:
  - Does not contain a record.
  - Is an after-image table with no associated before-image table.
- You can invoke the ROW-STATE function from within a WHERE clause (unlike the corresponding attribute). For example:

```
WHERE ROW-STATE(ttOrder) = ROW-MODIFIED.
```

See also Buffer object handle. ROW-STATE attribute, TRACKING-CHANGES attribute

ROWID function

Returns the unique internal identifier of the database record currently associated with the record buffer you name. This internal identifier has the data type ROWID, which is supported for Progress and all other DataServer databases.

Note: The ROWID function corresponds to the ROWID attribute.

This function replaces the RECID function for most applications. However, you must use the RECID function for maintaining schema objects (file and field relationships) in the Progress metaschema files. For more information, see *OpenEdge Development: Progress 4GL Handbook*.

Syntax

```
ROWID ( record )
```

record

The name of the record whose ROWID you want.

To use the ROWID function with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

Example

You might decide that you do not want to lock a record until the user starts to update that record. In the example procedure, the FIND statement reads a customer record without locking the record. The ROWID function puts the internal database identifier of that record in the crowid variable. If the user decides to update the credit-limit field, the procedure finds the record again using the value in crowid. The second FIND statement reads the record again, this time placing an EXCLUSIVE-LOCK on it. Because the record is first found with NO-LOCK, it is possible for the record to be updated by another user after the first FIND and before the second.

r-rowid.p

```
DEFINE VARIABLE response AS LOGICAL.
DEFINE VARIABLE crowid AS ROWID.

REPEAT:
    PROMPT-FOR customer.cust-num.
    FIND customer USING cust-num NO-LOCK.
    crowid = ROWID(customer).
    DISPLAY name .
    response = YES.
    UPDATE response LABEL "Update credit-limit ?".
    IF response THEN DO:
        FIND customer WHERE ROWID(customer) = crowid EXCLUSIVE-LOCK.
        UPDATE credit-limit.
    END.
END.
```

Notes

- Use the ROWID function to rapidly retrieve a previously identified record, even if that record has no unique index.
- The ROWID data type is a variable-length byte string capable of representing a record identifier for any DataServer database. However, the scope of a specific ROWID returned by the ROWID function depends on the DataServer and possibly the table within a database. The ROWID values for some DataServers change whenever the corresponding record is modified. For others, a ROWID value can change when a particular column in a table is modified. For more information on how different DataServers derive and work with ROWID values, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.
- You cannot return a ROWID for a view because view records do not have unique identifiers.

- You can compare ROWID values using the Progress relational operators (=, >, <, <>, >=, and <=), such as in the WHERE option of the Record phrase.
- You can use a ROWID value in a REPOSITION statement to specify the new position for a query cursor.
- If you want a called procedure to use the same record as a calling procedure, use the ROWID function to ensure that you are retrieving the same record. Use a SHARED ROWID variable or procedure parameter to communicate the ROWID of a record from one procedure to another. The second procedure can then find the same record. This is an alternative to using shared buffers or buffer parameters.
- You can store a ROWID value in a work table, but not directly in a temporary table or database table. You can use the STRING function to convert a ROWID value to a character string, which you can store in a temporary or database table, and convert it back to a ROWID value using the TO-ROWID function.
- You do not have to explicitly check to see whether a record is AVAILABLE before using the ROWID function. The ROWID function returns the Unknown value (?) if a record cannot be accessed.

  This example checks the ROWID for each Customer record returned for a query to determine if another record exists to update. If no more records exist, the update loop (QuickFix) terminates.

```
OPEN QUERY custq FOR EACH Customer
    WHERE Balance > 5000 AND Balance < 6000.
QuickFix: REPEAT:
    GET NEXT custq.
    IF ROWID(Customer) = ?
    THEN LEAVE QuickFix.
    ELSE UPDATE Customer.
END. /* QuickFix */
```

See also DEFINE BUFFER statement, DEFINE VARIABLE statement, RECID function, Record phrase, REPOSITION statement, STRING function, TO-ROWID function

RUN statement

Calls a Progress procedure. This procedure can be local to or remote from the current session, external from or internal to the current procedure, and either synchronous or asynchronous. When a local or remote procedure is called synchronously, the calling procedure resumes execution only after the called procedure completes execution. When a remote procedure is called asynchronously, the calling procedure resumes execution immediately after the remote request is sent to the AppServer.

The RUN statement can also call functions or routines that reside in the Windows Dynamic Link Libraries (DLLs) or in UNIX shared libraries. The called routine must first be declared like a Progress internal procedure. The procedure declaration must be in the same file as the RUN statement.

You can also use the RUN statement to create and associate a procedure object with a Web service, and invoke a Web service operation.

Syntax

```
RUN
  {     extern-proc-name
      | VALUE ( extern-expression )
      | path-name<<member-name>>
  }
  [ PERSISTENT [ SET proc-handle ] ]
  [ ON [ SERVER ] { server-handle | session-handle }
       [ TRANSACTION DISTINCT ]
       [ ASYNCHRONOUS
            [ SET async-request-handle ]
            [ EVENT-PROCEDURE event-internal-procedure
                 [ IN procedure-context ] ]
       ]
  ]
  [ ( parameter [ , parameter ] ... ) ]
  [ argument ] ...
  [ NO-ERROR ]
```

```
RUN
  { intern-proc-name | VALUE ( intern-expression) }
  [ IN proc-handle ]
  [ ASYNCHRONOUS
       [ SET async-request-handle ]
       [ EVENT-PROCEDURE event-internal-procedure
           [ IN procedure-context ] ]
  ]
  [ ( parameter [ , parameter ] ... ) ]
  [ NO-ERROR ]
```

Use the following syntax to create and associate a procedure object with a Web service:

```
RUN portTypeName [ SET hPortType ] ON SERVER hWebService [ NO-ERROR ] .
```

Use the following syntax to invoke a Web service operation:

```
RUN operationName IN hPortType
  [ ASYNCHRONOUS
    [ SET async-request-handle ]
    [ EVENT-PROCEDURE event-internal-procedure
      [ IN procedure-context ] ]
  [ ( parameter [ , parameter ] ... ) ]
  [ NO-ERROR ].
```

extern-proc-name

The name of the (local or remote) external procedure to run. On UNIX, external procedure names are case sensitive; in Windows, they are not. If you specify a relative pathname, Progress searches the directories (and libraries, on platforms that support libraries) defined in the PROPATH environment variable. With *extern-proc-name*, you can specify a local or remote procedure.

VALUE (*extern-expression*)

An expression that returns the name of the (local or remote) external procedure you want to run.

path-name<<member-name>>

The pathname of an r-code library and the name of an r-code file in that library. To specify an r-code file in a library, you must use the double angle brackets as shown. If you specify a relative library pathname, Progress searches the libraries defined in the PROPATH environment variable.

PERSISTENT [SET *proc-handle*]

Specifies that the external procedure be run and created (instantiated) as a persistent procedure. You can return the handle to the persistent procedure in *proc-handle* , a field, variable, or output parameter defined with the HANDLE data type. If you do not specify *proc-handle* , you can find the procedure handle for this procedure using the FIRST-PROCEDURE and LAST-PROCEDURE attributes of the SESSION system handle. You can use PERSIST as an abbreviation for PERSISTENT.

A persistent procedure creates and maintains its context after it returns to the caller. Other external procedures can access this context through procedure triggers and internal procedures defined in the persistent procedure. Thus, a RUN statement that runs and creates a persistent procedure context is referred to as an *instantiating RUN statement*.

The order of the PERSISTENT option and the ON SERVER option is interchangeable.

ON [SERVER] *server-handle*

Tells Progress to run the procedure remotely in the AppServer that the HANDLE variable, *server-handle*, refers to.

With the ASYNCHRONOUS option, *server-handle* causes the called procedure to run **asynchronously** in the remote session. Control returns immediately to the statement following the RUN statement. Execution of any specified *event-internal-procedure*. occurs in the context of an I/O blocking or PROCESS EVENTS statement.

The order of the PERSISTENT option and the ON SERVER option is interchangeable.

ON [SERVER] *session-handle*

Tells Progress to run the procedure locally in the current OpenEdge session, specified by the value of the SESSION system handle (*session-handle*).

With the ASYNCHRONOUS option, *session-handle* causes the called procedure to run **synchronously** in the local session, followed immediately by execution of any specified *event-internal-procedure*. Only after execution of the specified *event-internal-procedure* does control return to the statement following the RUN statement.

Note: This order of execution is different than for a remote procedure call using the *server-handle*.

The order of the PERSISTENT option and the ON SERVER option is interchangeable.

TRANSACTION DISTINCT

Tells Progress not to propagate the calling procedure's transaction to the AppServer. Although the current version of Progress does not allow transaction propagation, future versions might. Thus, to accommodate this possibility without breaking current code, the current version of Progress allows you to specify this option with *server-handle*.

Note: It is an error to specify TRANSACTION DISTINCT with a *session-handle*.

ASYNCHRONOUS [SET *async-request-handle*]

Specifies that the remote procedure is to be called as an asynchronous request. By default, the remote procedure is called synchronously. The handle to the asynchronous request is returned in *async-request-handle*, which must be a field, variable, or parameter defined with the HANDLE data type. If you specify ASYNCHRONOUS but do not specify SET *async-request-handle*, you can find the handle for the asynchronous request using the LAST-ASYNC-REQUEST attribute of the *server-handle* specified by the ON option. You can also locate the asynchronous request handle by walking the chain between the FIRST-ASYNC-REQUEST and LAST-ASYNC-REQUEST attributes of *server-handle*, searching on the PROCEDURE-NAME attribute of each request handle.

For a Web service operation invoked asynchronously, the handle that is set to the asynchronous request object created for the asynchronous request.

EVENT-PROCEDURE *event-internal-procedure*

Specifies a quoted string or character expression representing the name of an internal procedure that resides within *procedure-context*. When the response from the asynchronous request is received (that is, a PROCEDURE-COMPLETE event occurs), the specified internal procedure is called during subsequent execution of a PROCESS EVENTS or I/O-blocking statement (such as WAIT-FOR). The specified *event-internal-procedure* processes any parameters and errors returned from the asynchronous request. If not specified, no event procedure is executed when the PROCEDURE-COMPLETE event occurs for the asynchronous request.

For information on how the *event-internal-procedure* handles parameters from the asynchronous request, see the *parameter* option. For information on how the *event-internal-procedure* handles errors from the asynchronous request, see the NO-ERROR option.

IN *procedure-context*

A handle to an active procedure that contains the internal procedure specified by *event-internal-procedure*. If not specified, THIS-PROCEDURE is used as the *procedure-context* value.

(*parameter* **[** , *parameter* **]** **...**)

Specifies one or more parameters to pass to the called procedure.

For the parameter passing syntax, see the Parameter passing syntax reference entry in this book.

Parameters must be defined in the called procedure. (See the DEFINE PARAMETER statement reference entry.) They must be passed in the same order as they are defined, and they must have compatible data types. Progress attempts to convert values for data types that do not match. If Progress cannot convert the value for a mismatched data type, the RUN statement fails with an error condition.

For OUTPUT parameters of an asynchronous remote procedure call only, you can specify *parameter-name* AS *data-type* as a prototype. The *parameter-name* is an arbitrary place-holder name and *data-type* must specify the Progress data type of the corresponding OUTPUT parameter in the asynchronous remote procedure. You can also specify OUTPUT parameters for an asynchronous remote procedure using a local *field*, *variable*, or TABLE *temp-table-name*. However, note that the asynchronous remote procedure does not return any values to OUTPUT or INPUT-OUTPUT parameters on the

RUN statement. These parameters are place holders only for values returned by the remote procedure to the specified *event-internal-procedure*.

Any specified *event-internal-procedure* can define only INPUT parameters and must define one INPUT parameter for each OUTPUT or INPUT-OUTPUT parameter defined in the asynchronous remote procedure. Each *event-internal-procedure* INPUT parameter must match the corresponding remote procedure OUTPUT or INPUT-OUTPUT parameter in order and data type. (As with other procedures, Progress attempts to convert the values for data types that do not match.) The asynchronous remote procedure returns the values of these parameters to the INPUT parameters of the *event-internal-procedure* after the remote procedure completes execution and the client session processes the associated PROCEDURE-COMPLETE event.

If you are running an internal procedure declared as a Windows dynamic link library (DLL) or UNIX shared library routine, you must match any RETURN parameter specified by a DEFINE PARAMETER statement with a corresponding OUTPUT parameter in the RUN statement. If the internal procedure does not specify the RETURN parameter, do not specify the corresponding OUTPUT parameter in the RUN statement.

For external procedures, the parenthesized list of run-time parameters must precede any compile-time arguments.

argument

A constant, field name, variable name, or expression that you want to pass as a compile-time argument to the external procedure you are running.

When you pass arguments to an external procedure, Progress converts those arguments to character format. Progress recompiles the called procedure, substitutes arguments, and then runs the procedure. You cannot precompile a procedure to which you pass arguments. (If you use shared variables instead of arguments, the procedure can be precompiled. This yields more efficient code.)

Note: You cannot pass compile-time arguments in a call to an internal procedure.

NO-ERROR

Specifies that any ERROR conditions that occur in the attempt to run the procedure are suppressed. This does **not** mean that all errors produced by the called procedure are suppressed; only errors caused by the RUN statement itself. Also, if a specified local or synchronous remote procedure performs a RETURN ERROR, an ERROR is raised for the RUN statement. After the RUN statement completes, you can check the ERROR-STATUS system handle for information on any errors that occurred.

For an asynchronous remote procedure, the result depends on where the errors occur. If the errors occur during the send phase of the asynchronous request, this raises the ERROR condition on the RUN statement in the client (which you can suppress with NO-ERROR). If the errors occur during execution of the remote request and are returned by the AppServer, this results in an implied NO-ERROR on the RUN statement, and you must check the ERROR-STATUS system handle as well as the attributes of the asynchronous request handle (*async-request-handle*) for any error returns in the specified *event-internal-procedure*. If the asynchronous remote procedure returns an unhandled STOP condition, ERROR-STATUS:ERROR and *async-request-handle*:ERROR are both set to FALSE and *async-request-handle*:STOP is set to TRUE.

The RUN statement returns ERROR or STOP for a variety of events depending on the type of procedure that is executed, which includes any of the following:

- All types of procedures
- Local procedures
- All remote procedures
- Synchronous remote procedures
- Asynchronous remote procedures

Table 45 summarizes when Progress raises ERROR or STOP for each type of procedure.

Table 45: RUN statement ERROR and STOP conditions *(1 of 2)*

Procedure type	Condition	Event
All procedures	ERROR	The run-time parameters are not compatible.
	ERROR	Any specified IN *proc-handle* option is invalid.
	ERROR	A called internal procedure is not found in the specified external procedure.
	ERROR	The procedure returns ERROR.
	STOP	The procedure returns an unhandled STOP.
Local procedures	STOP	The specified procedure is not found.[1]
	STOP	An attempted compile of the procedure failed.[1]

Table 45: RUN statement ERROR and STOP conditions *(2 of 2)*

Procedure type	Condition	Event
All remote procedures	ERROR	The specified procedure is not found.
	ERROR	An attempted compile of the procedure failed.
	ERROR	The specified ON SERVER *server-handle* option is invalid.
	ERROR	The *server-handle* is not currently connected to some AppServer.
	ERROR	One of the parameters specified by *parameter* has a data type of BUFFER or MEMPTR.
	ERROR	The PROXY attribute of *proc-handle* (from the IN *proc-handle* option) is TRUE and the associated server handle is no longer connected to an AppServer.
Synchronous remote procedures	ERROR	The ASYNC-REQUEST-COUNT attribute on the *server-handle* is greater than zero (0).
Asynchronous remote procedures	ERROR	The REMOTE attribute of *procedure-context* is set to TRUE.

[1] The STOP condition, in this case, is supported for backward compatibility.

In addition, under the following conditions, a STOP condition occurs in the context of the I/O-blocking or PROCESS EVENTS statement that invokes any specified *event-internal-procedure*:

- Progress cannot locate the specified *event-internal-procedure*, for example, because the spelling of *event-internal-procedure* is not identical to the name of the internal procedure definition intended for use as the event procedure.
- The procedure handle that specifies the *procedure-context* to contain the definition of *event-internal-procedure* is not a valid procedure handle.

intern-proc-name

The name of the (local or remote) internal procedure you want to run. The procedure must be declared in the same procedure file as the RUN statement that calls it unless you specify the IN *proc-handle* option or use a super procedure. If you do not specify the IN *proc-handle* option and there is no internal procedure declared by the specified name, Progress tries to run an external procedure with the specified name. If the internal procedure is remote, you must specify the IN *proc-handle* option to identify the remote persistent procedure that defines the internal procedure on an AppServer.

VALUE (*intern-expression*)

An expression that evaluates to the name of the internal procedure you want to run.

IN *proc-handle*

Specifies the handle of the external procedure that declares the internal procedure you want to run. You can specify *proc-handle* as a field, variable, parameter, or expression that specifies a valid procedure handle or proxy (remote) persistent procedure handle.

portTypeName

The name of a Web service PortType as specified in the WSDL file.

hPortType

A handle to a procedure object that encapsulates a Web service operation.

hWebService

A handle to the server object bound to the Web service.

operationName

The name of a Web service operation specified in a WSDL file.

Examples

The following procedure displays a simple menu. The user's selection is stored in the selection variable. The INDEX function returns an integer value that indicates the position of the user's selection in a string of characters ("12345"). If the value in the selection variable is not in the list of values, the INDEX function returns a 0. The VALIDATE statement ensures that the INDEX function did not return a zero. If it did, VALIDATE displays the message "Not a valid choice."

r-run.p

```
DEFINE VARIABLE selection AS CHARACTER
LABEL "Enter Program Choice" FORMAT "x(1)".
DEFINE VARIABLE programs AS CHARACTER
FORMAT "x(15)" EXTENT 5.

/* Create the procedures custrpt.p, custedit.p, ordrpt.p, and ordedit.p.*/
programs[1] = "custrpt.p".
programs[2] = "custedit.p".
programs[3] = "ordrpt.p".
programs[4] = "ordedit.p".
programs[5] = "r-exit.p".

REPEAT:
  FORM HEADER TODAY "MASTER MENU" AT 35 STRING(TIME,"hh:mm") to 79.
  FORM SKIP(3)
   "1 - Customer Listing"  AT 30
   "2 - Customer Update" AT 30
   "3 - Order Listing"  AT 30
   "4 - Order Update"   AT 30
   "5 - Quit System"    AT 30
   selection COLON 28 AUTO-RETURN WITH SIDE-LABELS NO-BOX 1 DOWN.

UPDATE selection
  VALIDATE(INDEX("12345",selection) NE 0,
      "Not a valid choice").
  HIDE ALL.
  RUN VALUE(programs[INDEX("12345",selection)]).
END.
```

In the RUN statement, the INDEX function returns the position of the user's selection in a character string. Suppose you chose option 2 from the menu. That option occupies the second position in the "12345" character string. Therefore, the INDEX function returns the number two (2). Using this number, the RUN statement reads, RUN VALUE(programs[2]). According to the assignments at the top of the procedure, the value of programs[2] is custedit.p. Now the RUN statement reads, RUN custedit.p, and the `r-run.p` procedure runs the `custedit.p` procedure.

The following two external procedures, `r-runper.p` and `r-perprc.p`, illustrate the PERSISTENT and IN *proc-handle* options of the RUN statement. The first procedure, a non-persistent control procedure, sets up a window to run and manage the second procedure as a persistent procedure.

r-runper.p

```
DEFINE VARIABLE phand AS HANDLE.
DEFINE VARIABLE nhand AS HANDLE.
DEFINE VARIABLE whand AS WIDGET-HANDLE.
DEFINE BUTTON bStart LABEL "Start Customer Query".
DEFINE BUTTON bRecall LABEL "Recall All Hidden Queries".
DEFINE BUTTON bExit LABEL "Exit".
DEFINE FRAME ControlFrame SKIP (.5)
    SPACE (2) bStart bRecall bExit SPACE (2) SKIP (.5).

ON CHOOSE OF bStart IN FRAME ControlFrame RUN r-perprc.p PERSISTENT.

ON CHOOSE OF bRecall IN FRAME ControlFrame DO:
    phand = SESSION:FIRST-PROCEDURE.
    DO WHILE VALID-HANDLE(phand):
        IF phand:PRIVATE-DATA = "Customer Browse" THEN
            RUN recall-query IN phand.
        phand = phand:NEXT-SIBLING.
    END.
END.

ON CHOOSE OF bExit IN FRAME ControlFrame DO:
    phand = SESSION:FIRST-PROCEDURE.
    DO WHILE VALID-HANDLE(phand):
        nhand = phand:NEXT-SIBLING.
        IF phand:PRIVATE-DATA = "Customer Browse" THEN
            RUN destroy-query IN phand.
        phand = nhand.
    END.
    APPLY "RETURN" TO THIS-PROCEDURE.
END.

SESSION:SYSTEM-ALERT-BOXES = TRUE.
CREATE WINDOW whand
    ASSIGN
        TITLE = "Customer Query Control"
        SCROLL-BARS = FALSE
        MESSAGE-AREA = FALSE
        MAX-HEIGHT-CHARS = FRAME ControlFrame:HEIGHT-CHARS
        MAX-WIDTH-CHARS = FRAME ControlFrame:WIDTH-CHARS.
CURRENT-WINDOW = whand.
ENABLE ALL WITH FRAME ControlFrame.
WAIT-FOR RETURN OF THIS-PROCEDURE.
```

r-perprc.p *(1 of 2)*

```
DEFINE QUERY custq FOR customer.
DEFINE BROWSE custb QUERY custq
    DISPLAY name balance credit-limit phone WITH 10 DOWN.
DEFINE BUTTON bName LABEL "Query on Name".
DEFINE BUTTON bBalance LABEL "Query on Balance".
DEFINE BUTTON bCredit LABEL "Query on Credit".
DEFINE BUTTON bHide LABEL "Hide Query".
DEFINE BUTTON bCancel LABEL "Cancel".

DEFINE FRAME CustFrame custb SKIP
    bName bBalance bCredit bHide bCancel.

DEFINE VARIABLE custwin AS WIDGET-HANDLE.

ON CHOOSE OF bName IN FRAME CustFrame DO:
    custwin:TITLE = "Customers by Name".
    OPEN QUERY custq FOR EACH customer BY name.
END.

ON CHOOSE OF bBalance IN FRAME CustFrame DO:
    custwin:TITLE = "Customers by Balance".
    OPEN QUERY custq FOR EACH customer BY balance DESCENDING.
END.

ON CHOOSE OF bCredit IN FRAME CustFrame DO:
    custwin:TITLE = "Customers by Credit".
    OPEN QUERY custq FOR EACH customer BY credit-limit DESCENDING.
END.

ON VALUE-CHANGED OF BROWSE custb DO:
    IF customer.balance >= (customer.credit-limit * 0.75) THEN DO:
        BELL.
        MESSAGE "Evaluate" customer.name "for credit increase.".
    END.
END.

IF THIS-PROCEDURE:PERSISTENT THEN DO:
    THIS-PROCEDURE:PRIVATE-DATA = "Customer Browse".
    CREATE WIDGET-POOL.
END.

CREATE WINDOW custwin
    ASSIGN
        TITLE = "Customer Browser"
        SCROLL-BARS = FALSE
        MAX-HEIGHT-CHARS = FRAME CustFrame:HEIGHT-CHARS
        MAX-WIDTH-CHARS = FRAME CustFrame:WIDTH-CHARS.
```

r-perprc.p *(2 of 2)*

```
THIS-PROCEDURE:CURRENT-WINDOW = custwin.

ENABLE ALL WITH FRAME CustFrame.

IF THIS-PROCEDURE:PERSISTENT THEN DO:
    ON CHOOSE OF bCancel IN FRAME CustFrame DO:
        RUN destroy-query.
    END.
    ON CHOOSE OF bHide IN FRAME CustFrame DO:
        custwin:VISIBLE = FALSE.
    END.
END.
ELSE DO:
    WAIT-FOR CHOOSE OF bHide, bCancel IN FRAME CustFrame.
END.

PROCEDURE recall-query:
    custwin:VISIBLE = TRUE.
END.

PROCEDURE destroy-query:
    DELETE PROCEDURE THIS-PROCEDURE NO-ERROR.
    DELETE WIDGET-POOL.
END.
```

The control procedure, `r-runper.p`, runs `r-perprc.p` each time you choose the Start Customer Query button. Each time it runs, `r-perprc.p` creates (instantiates) an additional context instance for the persistent procedure, including an additional window to open customer queries. When you choose the Recall All Hidden Queries button from the control window, `r-runper.p` calls the recall-query internal procedure in each instance of `r-perprc.p` to redisplay its window. Similarly, when you choose the Exit button, `r-runper.p` calls the destroy-query internal procedure in each instance of `r-perprc.p` to delete its context instance; `r-runper.p` then applies the RETURN event to itself to terminate by completing the WAIT-FOR statement.

The `r-perprc.p` procedure sets up a customer query that you can re-open three different ways: by name, by balance, or by credit. Each instance of `r-perprc.p` maintains a separate query for its own local customer buffer. Note that by testing and setting attributes of the THIS-PROCEDURE system handle, `r-perprc.p` can run either persistently or non-persistently. The basic difference is how the procedure maintains its own context. For example, when running persistently, it defines a trigger on the bCancel button to run its own deletion procedure, destroy-query, to terminate; when running non-persistently, it completes a WAIT-FOR statement with the bCancel button to terminate.

The following example shows how you might implement an asynchronous request. The procedure `r-async.p` runs persistently from a user-interface trigger, perhaps in response to a menu choice. This procedure, in turn, sends a request to run `runReport.p` on an AppServer, which provides an inventory report for the specified date.

When `r-async.p` returns, the user-interface trigger ends and the application returns to its WAIT-FOR state. The user continues to use the application in the normal way while the inventory report runs on the AppServer.

When `runReport.p` finishes running, a PROCEDURE-COMPLETE event occurs. This event causes the internal procedure reportDone to run automatically within the context of the application's WAIT-FOR statement. Whatever the user is doing in the application, reportDone displays an alert box indicating whether or not the inventory report completed successfully and the number of lines (numLines) that were output for the report. (The bolded 4GL indicates the code required to support asynchronous requests to run `runReport.p`.)

r-async.p

```
DEFINE INPUT PARAMETER invDate AS DATE.
DEFINE VAR sh AS WIDGET-HANDLE. /* Server handle */
DEFINE VAR ah AS WIDGET-HANDLE. /* Asynchronous request handle */

CREATE SERVER sh.
sh:CONNECT("-AppService Inventory -H myhost").
RUN runReport.p
    ON SERVER sh
    ASYNCHRONOUS SET ah EVENT-PROCEDURE "reportDone" IN THIS-PROCEDURE
    (invDate, OUTPUT numLines AS INT).
RETURN.

PROCEDURE reportDone:
    DEFINE INPUT PARAMETER numLines AS INT.
    IF ah:ERROR OR ah:STOP THEN DO:
        MESSAGE "An error occurred when running your" SKIP
                "Inventory report for " invDate "." SKIP
                "The error is: " ERROR-STATUS:GET-MESSAGE(1)
                VIEW-AS ALERT-BOX.
    END.
    ELSE DO:
        MESSAGE "Your Inventory report for " invDate SKIP
                "has completed successfully." SKIP
                numLines " report lines were generated"
                VIEW-AS ALERT-BOX.
    END.
    sh:DISCONNECT().
    DELETE OBJECT sh.
    DELETE OBJECT THIS-PROCEDURE. /* Persistent proc no longer needed */
END.
```

Notes

- 4GL procedures can be run recursively (a procedure can run itself).
- In Version 6, Progress uses time stamps by default to verify that r-code is consistent with the database schema. Some releases of Version 6 provide optional support for CRC codes instead of time stamps. Progress Version 7 and later uses CRC codes by default. If you want to use time stamps instead, specify the Time Stamp (-tstamp) parameter when you connect to a database.
- When a RUN statement raises the STOP condition, Progress displays the resulting messages on the current output device, even if you specify NO-ERROR. Progress also writes these messages to the ERROR-STATUS system handle, but sets ERROR-STATUS:ERROR to FALSE.
- You can run an internal procedure that is declared in the current external procedure or in the procedure you specify with the IN *proc-handle* option. The procedure handle specified by the IN *proc-handle* option can specify either a valid persistent procedure instance or an external procedure that is active on the procedure call stack. The handle can also specify the current external procedure using the THIS-PROCEDURE system handle. You can check the validity of any procedure handle using the VALID-HANDLE function.
- A called external procedure uses any arguments passed to it from the calling procedure by referring to those arguments as numbers enclosed in braces { }. The first argument is {1}, the next is {2}, etc. Any arguments the called procedure does not use are ignored, and any missing arguments are treated as null values. (Note that the null is a legal value in a WHERE or WITH clause, but its occurrence can cause an error at other points in a called procedure.)
- To run an r-code file stored in a library that is not on PROPATH, you must specify the name of the library and the name of the r-code file in the library. Specify these names in the form *path-name*<<*member-name*>>, where *path-name* is the pathname of the library and *member-name* is the name of the r-code file. For example, if you have an r-code file called appmenu.r in a library whose pathname is /usr/foo/app.pl, you use this command to run it:

```
RUN /usr/foo/app.pl<<appmenu.r>>.
```

- When you run a procedure and do not specify the PERSISTENT option, Progress first looks for an internal procedure with the name you specify (this search is not case sensitive). If you specify a procedure in the form *path-name<<member-name>>*, Progress looks for an internal procedure with a name in that form. If you specify the PERSISTENT option, or if no internal procedure is found, Progress searches all the directories and libraries in PROPATH for a usable r-code file of the same name. Progress also checks to see if the procedure was modified since the last time it was run. If there is a usable r-code file, there is no point in performing the compilation. The RUN statement always uses an existing r-code file before using a session compile version of a procedure.

  If you do not want Progress to check whether the procedure has been modified before using the r-code, use the Quick Request (-q) parameter.

- When running an external procedure, it is good practice to specify the name of the source file in the RUN statement. For example, to run r-exit.p you specify the following:

  ```
  RUN r-exit.p
  ```

  When you specify a suffix or file extension (such as .p), Progress first tries replacing that suffix or extension with .r and searches the first directory on your PROPATH for a file with that name. If the r-code file is not found, then it reverts to the original suffix and searches for a source file with that name. If the source file is not found in the first PROPATH directory, then Progress searches for an r-code file and then a source file in each subsequent directory on your PROPATH until a file is found.

  If you specify the .r suffix in the RUN statement, then Progress searches only for an r-code file in each directory on your PROPATH. If you omit the extension, then Progress first adds a .r to the name you specify and searches the first directory for an r-code file with that name. If none is found, then Progress searches for a source file with no suffix or extension.

- You cannot run an internal procedure with the PERSISTENT option.

- An external procedure called with the PERSISTENT option runs in the same way as a non-persistent procedure with these differences:
  - The procedure does not go out of scope when it returns: its context and most of its allocated resources remain active, including input parameters, widgets, variables, buffers, temporary tables, work tables, and triggers created during procedure execution. However, all **static** dialog boxes, their child widgets, and related triggers created during its execution are destroyed when the procedure returns to the caller. This makes all other windows and dialog boxes in the application available for input.
  - All buffers passed as parameters to a persistent procedure are treated as local buffers in the persistent context. When the procedure instantiation returns, the output value of the buffer parameter is returned, as usual, to the calling procedure. However, any cursor positioning established during execution of the instantiating RUN statement is lost to the persistent context once the procedure returns; Progress creates a copy of the buffer parameter and resets its cursors as an initially defined local buffer.
  - If the procedure obtains any schema share locks (through database access) while executing, these remain in effect after the procedure returns, until the procedure is deleted.
  - Each time you run a procedure persistently, you create a new instance of its procedure context. All of its data, buffers, and widgets are duplicated and separately managed by the new instantiation until the procedure instance is deleted.

Note: If you run an application that creates persistent procedures from an ADE tool (for example, the Procedure Editor or User Interface Builder), that tool removes all instances of persistent procedures still created when the application terminates.

- Transaction scoping is the same whether you run a procedure as persistent or not. Any transaction which begins inside a persistent procedure is scoped to the block that starts the transaction.
- If you run a procedure with the PERSISTENT option and a STOP or QUIT condition or a RETURN ERROR occurs during execution of the procedure, the procedure returns as a non-persistent procedure.
- All shared variables, buffers, temporary tables, ProDataSet objects, work tables, and queries remain in scope as long as a persistent procedure instance remains that accesses them. This is true even if the procedure (persistent or non-persistent) that originally defined the shared data has gone out of scope. Shared data can go out of scope only when no persistent procedure remains that references it.

- You cannot run a procedure with the PERSISTENT option in which you have defined shared streams or shared frame, browse, or menu widgets. Doing so causes Progress to raise ERROR on the RUN statement.

- You can remove an instance of a persistent procedure using the DELETE PROCEDURE statement. When you delete the procedure instance, its context goes out of scope and all allocated resources are returned to the system. If the procedure has shared dependencies on the call stack, the delete pends until the dependencies are cleared.

- To run a Windows DLL routine as an internal procedure, you must reference the DLL in a PROCEDURE statement and define its parameters in the associated internal procedure block. For more information on accessing DLL routines from Progress, see *OpenEdge Development: Programming Interfaces* manual.

- To run a UNIX shared library routine as an internal procedure, you must reference the UNIX shared library in a PROCEDURE statement and define its parameters in the associated internal procedure block. You can declare an internal procedure as a routine in a UNIX shared library in the same manner as declaring a DLL routine. The one exception is that the ORDINAL option is not applicable to UNIX and will be ignored.

```
RUN atoi(INPUT in-string, OUTPUT out-int).
```

- You can define triggers on procedure handles (procedure triggers). You can apply events to any procedure trigger defined either within a persistent procedure or within any external procedure that is active on the procedure call stack.

```
DEFINE VARIABLE phand AS HANDLE.
RUN persproc.p PERSISTENT SET phand.
               .
               .
               .
APPLY "RETURN" TO phand.
```

This code fragment assumes that a trigger is defined within `persproc.p` for the RETURN event on the THIS-PROCEDURE handle. For more information on defining and executing procedure triggers, see *OpenEdge Development: Progress 4GL Handbook*.

- If you are using Progress with a DataServer that supports stored procedures, the RUN statement has extensions that allow you to execute a stored procedure. For more information, see the entry for the RUN STORED-PROCEDURE statement and the appropriate OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

- If you RUN a procedure multiple times within a session, changing the procedure between runs, you must manually recompile the procedure each time. Otherwise, the procedure's last r-code, which persists for a session, is what is run and the changes do not appear.

- An asynchronous call to a remote procedure (using the ASYNCHRONOUS option) causes the RUN statement to return control immediately to the following statement in the local context, whether or not the remote procedure has completed execution.

- If an asynchronous call to a remote procedure **does not** raise a STOP or ERROR condition, Progress:

  – Increments the *server-handle*:ASYNC-REQUEST-COUNT attribute.

  – Increments the *proc-handle*:ASYNC-REQUEST-COUNT attribute, if PERSISTENT is specified for a remote external procedure or IN *proc-handle* is specified for a remote internal procedure.

  – Sets the *async-request-handle*:COMPLETE attribute to FALSE, indicating that the request has not completed execution.

  – Sets the *async-request-handle*:EVENT-PROCEDURE attribute to the value of *event-internal-procedure*, if *event-internal-procedure* is specified.

  – Sets the *async-request-handle*:EVENT-PROCEDURE-CONTEXT attribute to the value of *procedure-context*, if *procedure-context* is specified.

  – Submits the request for execution by the AppServer.

- Progress checks the syntax of the ON SERVER option at run time. This allows you to use a single HANDLE variable that you can set either to a server handle value or the value of the current SESSION handle. Thus, you can use the same RUN statement to execute a procedure remotely in an AppServer or locally depending on application conditions.

- When you specify the ON SERVER option with the SESSION system handle, the RUN statement is functionally similar to not specifying the ON SERVER option at all. That is, the two RUN statements in the following code perform the same function:

```
DEFINE VARIABLE hServer AS HANDLE.
hServer = SESSION.
RUN foo.p.
RUN foo.p ON SERVER hServer.
```

  Allowing the same ON SERVER option to specify either a local session or a remote AppServer session facilitates code generation for applications like the Progress AppBuilder.

  With the ASYNCHRONOUS option, using the ON SERVER SESSION option causes the called procedure to run **synchronously** in the local session, followed immediately by execution of any specified *event-internal-procedure*. Only after execution of the specified *event-internal-procedure* does control return to the statement following the RUN statement. This synchronous local execution includes the following differences in error handling from asynchronous execution on an AppServer using ON SERVER *server-handle*:

  – If an unhandled ERROR condition occurs during execution of the called local procedure, the error message is displayed on the local output device. This is different from remote execution, where any error message is written to the AppServer log file.

  – If the called local procedure causes an ERROR or STOP condition to be raised in the calling procedure (a file not found, mismatched parameters, a compile error, and explicit execution of a RETURN ERROR or STOP statement), Progress sends the associated message to the standard output device and sets ERROR-STATUS:ERROR appropriately. This is different from remote execution, where Progress in most cases attaches the associated message to the ERROR-STATUS system handle.

  – Also, if the called local procedure causes an ERROR or STOP condition to be raised in the calling procedure (as in the previous note), Progress raises the condition on the RUN statement, as for a local RUN statement without the ON SERVER option. This is different from remote execution, where Progress does not raise the condition on the calling RUN statement. You can work around this for the ON SERVER SESSION case by coding each asynchronous RUN statement with the NO-ERROR option and possibly surrounding it with a DO ON STOP UNDO, LEAVE block.

- For more information on AppServers and calling remote procedures synchronously or asynchronously, see *OpenEdge Application Server: Developing AppServer Applications*.
- For more information on asynchronous invocation of Web service operations, see *OpenEdge Development: Web Services*.

See also

{ } Argument reference, { } Include file reference, APPLY statement, Asynchronous request object handle, CODEBASE-LOCATOR system handle, COMPILE statement, CREATE SAX-READER statement, DEFINE PARAMETER statement, DELETE PROCEDURE statement, ON statement, Parameter passing syntax, PROCEDURE statement, RUN STORED-PROCEDURE statement, THIS-PROCEDURE system handle, VALID-HANDLE function, Widget phrase

RUN STORED-PROCEDURE statement

Runs a non-Progress stored procedure or allows you to send SQL to an SQL-based data source using an OpenEdge DataServer.

Syntax

```
RUN STORED-PROCEDURE procedure
  [ integer-field = PROC-HANDLE ]
  [ NO-ERROR ]
  [ ( parameter [ , parameter ] ... ) ]
```

procedure

The name of the stored procedure that you want to run or the Progress built-in procedure name, `send-sql-statement`, to send SQL to an SQL-based data source.

integer-field = PROC-HANDLE

Assigns a value to the specified integer field or variable (*integer-field*) that uniquely identifies the stored procedure returning results from the non-OpenEdge database or that uniquely identifies the SQL cursor used to retrieve results from an SQL-based, ODBC-compliant data source.

NO-ERROR

Specifies that any ERROR conditions that the RUN STORED-PROCEDURE statement produces are suppressed. Before you close a stored procedure, check the ERROR-STATUS handle for information on any errors that occurred. You receive an error when you attempt to close a stored procedure that did not start.

Note: This option must appear before any run-time parameter list.

parameter

A run-time parameter to be passed to the stored procedure. A *parameter* has the following syntax:

```
[ INPUT | OUTPUT | INPUT-OUTPUT ]
  [ PARAM parameter-name = ] expression
```

An *expression* is a constant, field name, variable name, or expression. INPUT is the default. OUTPUT and INPUT-OUTPUT parameters must be record fields or program variables. For ORACLE, OUTPUT and INPUT-OUTPUT work the same way.

If you run send-sql-statement for an SQL-based data source, you must pass a single character expression *parameter* containing the SQL statement you want the data source to execute.

If you do not specify *parameter-name* (the name of a keyword parameter defined by the stored procedure), you must supply all of the parameters in correct order. If you do specify *parameter-name*, you must precede your assignment statement with the keyword PARAM. If you do not supply a required parameter, and no default is specified in the stored procedure, you receive a run-time error.

Examples

This procedure runs the ORACLE stored procedure pcust and writes the results of the stored procedure into the Progress-supplied buffer, proc-text-buffer. The same code works for accessing a stored procedure from an ODBC-compliant data source.

```
DEFINE VAR intvar AS INTEGER.

RUN STORED-PROCEDURE pcust intvar = PROC-HANDLE
(10, OUTPUT 0, OUTPUT 0) NO-ERROR.
FOR EACH proc-text-buffer WHERE PROC-HANDLE = intvar:
  DISPLAY proc-text-buffer.
END.
IF ERROR-STATUS:ERROR
THEN DO:
  MESSAGE "Stored Procedure failed to run".
END.
ELSE CLOSE STORED-PROCEDURE pcust WHERE PROC-HANDLE = intvar.
```

This procedure uses the `send-sql-statement` option of the RUN STORED-PROCEDURE statement to send SQL to ORACLE. It writes the results of the stored procedure into the Progress-supplied buffer, proc-text-buffer. The same code works for sending SQL to an ODBC-compliant data source:

```
DEFINE VAR handle1 AS INTEGER.

RUN STORED-PROC send-sql-statement handle1 = PROC-HANDLE
     ("SELECT name, cust_num FROM customer").
FOR EACH proc-text-buffer WHERE PROC-HANDLE = handle1:
   display proc-text.
   END.
CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = handle1.
END.
```

This code example shows how to trap errors from the non-OpenEdge RDBMS within a procedure:

```
DEFINE VAR h1 AS INTEGER.
DEFINE VAR j AS INTEGER.
RUN STORED-PROC send-sql-statement h1 = PROC-HANDLE NO-ERROR
  ("select count (*) from xxx.customer where name between 'A' and 'Z' ").

IF ERROR STATUS:ERROR THEN DO:
  DO j = 1 TO ERROR-STATUS:NUM-MESSAGES:
    MESSAGE "error" ERROR-STATUS:GET-NUMBER(j)
        ERROR-STATUS:GET-MESSAGE(j).
  END.
END.

CLOSE STORED-PROC send-sql-statement WHERE PROC-HANDLE = h1.
```

Notes

- The RUN STORED-PROCEDURE statement starts a transaction with the same scope as transactions started with the UPDATE statement.
- For more information on using this statement and on using the built-in procedure name, `send-sql-statement`, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

See also

CLOSE STORED-PROCEDURE statement, PROC-HANDLE function, PROC-STATUS function

RUN SUPER statement

Runs the super procedure version of the current internal procedure.

The RUN SUPER statement must appear only within an internal procedure, but can appear anywhere within the internal procedure. If the RUN SUPER statement appears outside an internal procedure, the compiler reports an error.

Syntax

```
RUN SUPER [ ( parameter [ , parameter ] ... ) ] [ NO-ERROR ]
```

parameter

A parameter of the super procedure. The parameters of the super procedure must have the same *signature* (number of parameters, and type and mode of each) as the parameters of the current internal procedure. You can, however, adjust a parameter's value.

For the *parameter* syntax, see the Parameter passing syntax reference entry in this book.

NO-ERROR

Suppresses the display of the error message if the search for the super procedure version of the current internal procedure fails. The error is still generated and stored in the ERROR-STATUS handle.

Note: Specifying NO-ERROR does not shorten the search in any way.

If you do not specify the NO-ERROR option and the super procedure version of the internal procedure does not exist, Progress generates an error message:

```
Procedure prog.p name has no SUPER procedure with internal procedure name
```

Example

The following example consists of three procedure files: a main routine, a driver, and a third procedure file that becomes a super procedure of the driver.

The following main routine, procedure file `r-pomain.p`, runs the driver procedure persistently:

r-pomain.p

```
/* r-pomain.p */
DEFINE VARIABLE h AS HANDLE.
DEFINE VARIABLE a AS CHARACTER.
FUNCTION sample2 RETURNS CHARACTER (INPUT-OUTPUT a AS CHARACTER) IN h.

RUN r-podrvr.p PERSISTENT SET h.
RUN sample1 IN h (INPUT-OUTPUT a).
MESSAGE a VIEW-AS ALERT-BOX.
a = "".
MESSAGE sample2(a) VIEW-AS ALERT-BOX.
```

The following driver, procedure file r-podrvr.p, runs the third procedure file persistently, makes it a super procedure of itself, defines the internal procedure sample1, and defines the user-defined functions sample2, GetPartName, and SetPartName:

r-podrvr.p

```
/* r-podrvr.p */
FUNCTION SetPartName RETURNS INTEGER (INPUT a AS CHARACTER) FORWARD.
DEFINE VARIABLE h AS HANDLE.DEFINE VARIABLE localPartName AS CHARACTER.

/* Add a super procedure */
RUN r-posupr.p PERSISTENT SET h.
THIS-PROCEDURE:ADD-SUPER-PROCEDURE (h).
SetPartName("1998 Calendar").

PROCEDURE sample1:
    DEFINE INPUT-OUTPUT PARAMETER a AS CHARACTER.
    a = a + "proc: Part name is: ".
    /* Invoke procedure sample1 in the super procedure. */
    RUN SUPER (INPUT-OUTPUT a).
END PROCEDURE.

FUNCTION sample2 RETURNS CHARACTER (INPUT-OUTPUT a AS CHARACTER).
    a = a + "func: Part name is: ".
    /* Invoke function sample2 in the super procedure. */
    SUPER (INPUT-OUTPUT a).
    RETURN a.
END FUNCTION.

FUNCTION GetPartName RETURNS CHARACTER ():
    RETURN localPartName.
END FUNCTION.

FUNCTION SetPartName RETURNS INTEGER (INPUT partname AS CHARACTER):
    localPartName = partname.
END FUNCTION.
```

The following third procedure file, r-posupr.p, defines a new version of the internal procedure sample1 and a new version of the user-defined function sample2:

r-posupr.p

```
/* r-posupr.p */
DEFINE VARIABLE h AS HANDLE.
FUNCTION GetPartName RETURNS CHARACTER () IN H.

PROCEDURE sample1:
    DEFINE INPUT-OUTPUT PARAMETER a AS CHARACTER.
    h = TARGET-PROCEDURE.
    a = a + GetPartName().
    MESSAGE "TARGET-PROCEDURE is:" TARGET-PROCEDURE:FILE-NAME
        VIEW-AS ALERT-BOX.
    MESSAGE "SOURCE-PROCEDURE is:" SOURCE-PROCEDURE:FILE-NAME
        VIEW-AS ALERT-BOX.
END PROCEDURE.

FUNCTION SAMPLE2 RETURNS CHARACTER (INPUT-OUTPUT a AS CHARACTER):
    h = TARGET-PROCEDURE.
    a = a + GetPartName().
    RETURN a.
END.
```

To start the example, run r-pomain.p from the Procedure Editor.

Notes

- To run the super version of a user-defined function, use the SUPER function.
- For the rules that Progress uses to find the super procedure, see the reference entry for the ADD-SUPER-PROCEDURE( ) method in this book.
- For an overview of super procedures, see *OpenEdge Development: Progress 4GL Handbook*.

See also

ADD-SUPER-PROCEDURE( ) method, REMOVE-SUPER-PROCEDURE( ) method, SOURCE-PROCEDURE system handle, SUPER function, SUPER-PROCEDURES attribute, TARGET-PROCEDURE system handle

SAVE CACHE statement

Saves the schema cache of a database to an operating system file. Subsequent sessions can then share the same cache by using the Schema Cache File (-cache) parameter.

Syntax

```
SAVE CACHE
    { CURRENT | COMPLETE }
    { database-name | VALUE ( char-expr ) }
  TO
    { pathname |  VALUE ( char-expr )}
[ NO-ERROR ]
```

CURRENT

Specifies that only the portion of the schema cache that applies to referenced tables is saved to the file. By using this option you can tailor a small schema cache file for an application that does not use all the tables in the database.

COMPLETE

Specifies that the complete schema cache for the database is saved to the file. If you use this option, the client process builds a complete schema cache in memory including template records and all trigger information for every table in the database.

database-name

Specifies the literal logical name of a currently connected OpenEdge database.

pathname

Specifies the literal pathname of an operating system file to hold the schema cache.

VALUE (*char-expr*)

Returns the corresponding literal database name or pathname specified by the character expression in *char-expr*.

NO-ERROR

Specifies that any errors that occur in the attempt to save the schema cache file are suppressed. After the SAVE CACHE statement completes, you can check the ERROR-STATUS system handle for information on any errors that occurred.

Example This procedure saves the complete schema cache for each database that you specify in the current working directory, and displays any error messages associated with connecting or saving the cache:

r-schcsh.p

```
DEFINE VARIABLE db-name AS CHARACTER FORMAT "x(12)" INITIAL ?.
DEFINE VARIABLE icnt AS INTEGER.

DO WHILE db-name <> "":
    SET db-name LABEL "Database Name"
        WITH FRAME A SIDE-LABELS TITLE "Save Cache" VIEW-AS DIALOG-BOX.
    IF db-name <> "" THEN
        CONNECT VALUE(db-name) -1 NO-ERROR.
    ELSE
        LEAVE.
    IF NOT ERROR-STATUS:ERROR THEN DO:
        SAVE CACHE COMPLETE VALUE(db-name) to VALUE(db-name + ".csh")
            NO-ERROR.
        IF NOT ERROR-STATUS:ERROR THEN
            MESSAGE "Saved schema cache for"
                    db-name "in" db-name + ".csh.".
        ELSE DO:
            BELL.
            DO icnt = 1 TO ERROR-STATUS:NUM-MESSAGES:
                MESSAGE ERROR-STATUS:GET-MESSAGE(icnt)
                    VIEW-AS ALERT-BOX.
            END.
        END.
    END.
    ELSE DO:
        BELL.
        DO icnt = 1 TO ERROR-STATUS:NUM-MESSAGES:
            MESSAGE ERROR-STATUS:GET-MESSAGE(icnt)
                VIEW-AS ALERT-BOX.
        END.
    END.
    DISCONNECT VALUE(db-name) NO-ERROR.
END.
```

Notes

- The schema cache is saved to the file in a binary format that is portable across machines.
- For information on using an existing schema cache file, see *OpenEdge Data Management: Database Administration*. For information on the Schema Cache File (-cache) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- Any schema changes to the database make the saved cache invalid. If the schema cache file is invalid when Progress tries to access it, Progress displays a warning message, ignores the file, and reads the required schema cache from the database.
- To set up your database environment to use the CURRENT option, you only have to connect to the database and read from the tables that compose the schema you want to save. This is sufficient for the SAVE CACHE statement to save all parts of each table in the schema, including template records and trigger information. If you want to save a different subschema of the database, you must disconnect and then reconnect to the database before reading the tables for that subschema.
- For a DataServer, Progress saves the schema cache for the entire schema holder database. You cannot save the schema cache for a non-OpenEdge database separately. For more information on schema cache files for DataServers, see the OpenEdge DataServer Guides, *OpenEdge Data Management: DataServer for Microsoft SQL Server*, *OpenEdge Data Management: DataServer for ODBC*, and *OpenEdge Data Management: DataServer for ORACLE*.

See also CONNECT statement, ERROR-STATUS system handle

SCREEN-LINES function

Returns the number of lines you can use to display frames. This value omits the space used by the message area and status area.

Note: Does not apply to SpeedScript programming.

Syntax

```
SCREEN-LINES
```

Example

Here, a different number of customer records is displayed depending on the number returned by the SCREEN-LINES function:

r-scrnln.p

```
DEFINE VARIABLE nbrdown AS INTEGER.

IF SCREEN-LINES > 21
THEN nbrdown = 7.
ELSE nbrdown = 6.

FOR EACH customer WITH nbrdown DOWN:
  DISPLAY cust-num name address city state country.
END.
```

SCROLL statement

Moves data up or down in a frame with multiple rows. Use the SCROLL statement to scroll data up or down when you add or delete a line in a frame.

This statement is supported only for backward compatibility.

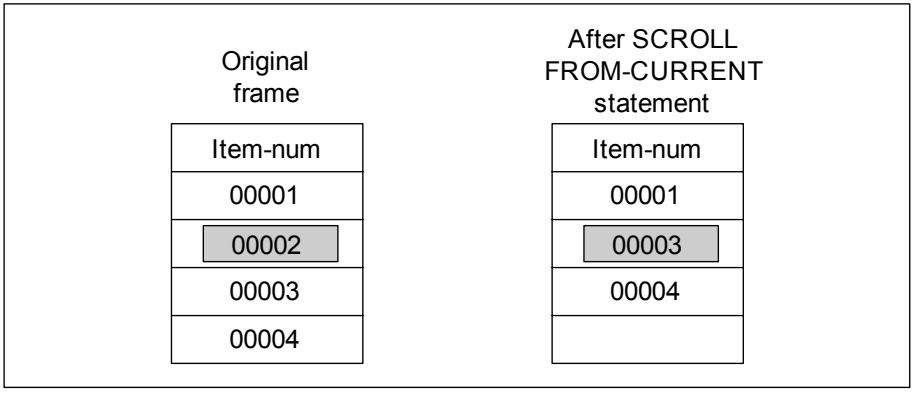
Note: Does not apply to SpeedScript programming.

Syntax

```
SCROLL
  [ FROM-CURRENT ]
  [ UP | DOWN ]
  { [ frame-phrase ] }
```

FROM-CURRENT

Scrolls UP or DOWN rows of data at or below the current cursor location. When scrolling UP, a new line opens at the bottom of the frame. When scrolling DOWN, a new line opens at the current cursor location. For example:

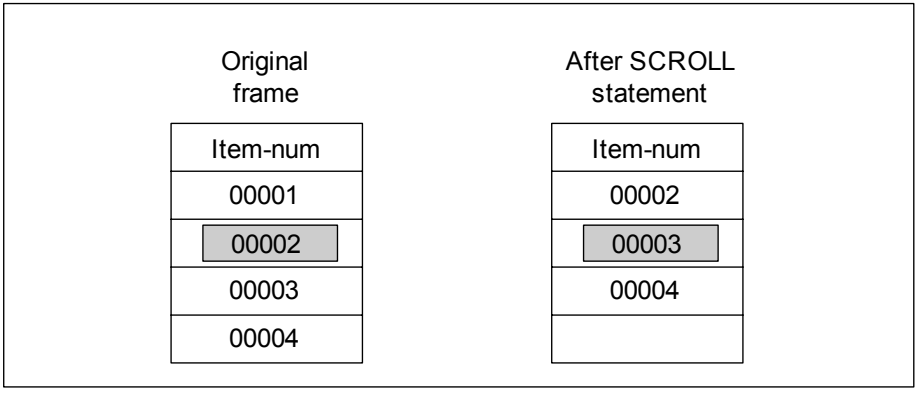


If you do not use the FROM-CURRENT option, then the entire frame scrolls up or down and the newly opened line appears at the top or bottom of a frame, respectively.

FROM-CURRENT limits scrolling from the current cursor position to the bottom of the frame.

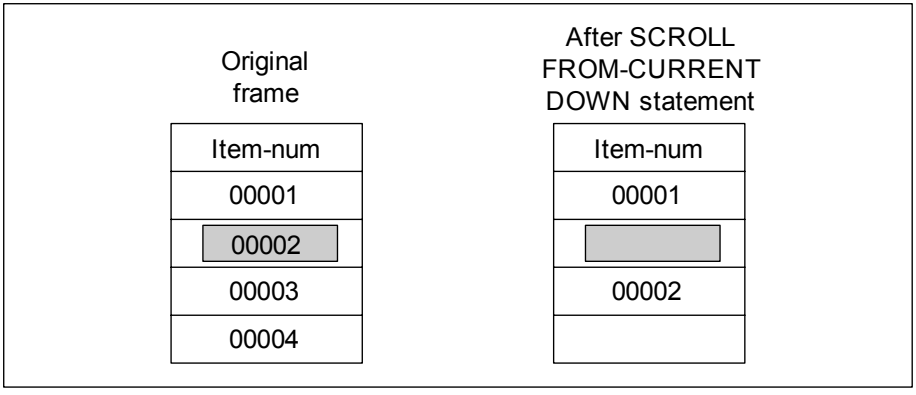
UP

Scrolls rows of data up and off the frame and opens a line at the bottom of the frame. UP is the default. For example:

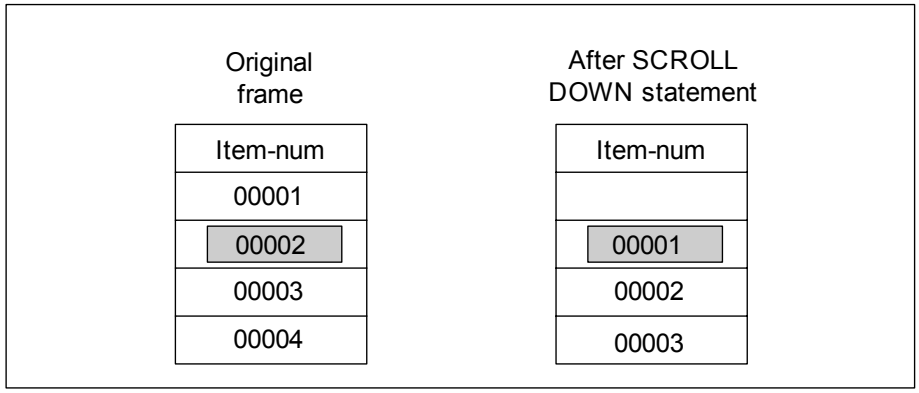


DOWN

Scrolls rows of data down and off the frame and opens a line at the top of the frame. For example, the Original Frame in the next example shows four rows of data. The highlighted bar is the current cursor position and the frame is a scrolling frame. On the right, the SCROLL FROM-CURRENT DOWN statement opens a line in the frame at the current cursor location and moves the other rows down and off the frame. For example:



In the next example, the SCROLL DOWN statement opens a line at the top of the frame and moves the other rows of data down and off the frame:



frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Examples This procedure displays customer information and lets you try each scrolling option from a menu of selections:

r-scroll.p

```
DEFINE VARIABLE ans AS CHARACTER FORMAT "x".

FORM cust.cust-num cust.name credit-limit
   WITH FRAME cust CENTERED 10 DOWN.

FORM  "1 - scroll up" SKIP
      "2 - scroll from-current up" SKIP
      "3 - scroll down" SKIP
      "4 - scroll from-current down" SKIP
      "5 - scroll from-current "
WITH FRAME instruct CENTERED TITLE "Instructions:".

VIEW FRAME cust.

REPEAT WHILE FRAME-LINE(cust) <= FRAME-DOWN(cust):
   FIND NEXT customer.
   DISPLAY cust-num name credit-limit WITH FRAME cust TITLE "Customers".
   DOWN WITH FRAME cust.
END.

UP FRAME-DOWN(cust) / 2 WITH FRAME cust.

VIEW FRAME instruct.REPEAT WITH FRAME cust:
   CHOOSE ROW cust.name KEYS ans AUTO-RETURN NO-ERROR WITH FRAME cust.
   IF ans = "1" THEN SCROLL UP.
   ELSE IF ans = "2" THEN SCROLL FROM-CURRENT UP.
   ELSE IF ans = "3" THEN SCROLL DOWN.
   ELSE IF ans = "4" THEN SCROLL FROM-CURRENT DOWN.
   ELSE IF ans = "5" THEN SCROLL FROM-CURRENT.
   VIEW FRAME cust.
   ans = "".
END.
```

The next procedure creates a scrolling frame of five iterations. The frame displays the `cust-num`, `name`, `address`, and `city` for each customer. The status default message displays “Enter C to create, D to delete” as long as the procedure is running. You use arrow keys to move the highlighted cursor bar through the database, and to add or delete customers from the database. The CHOOSE statement lets you easily create this style menu. See the CHOOSE statement reference entry for more information.

r-chose1.p *(1 of 3)*

```
DEFINE VARIABLE counter AS INTEGER.
DEFINE VARIABLE oldchoice AS CHARACTER.

FORM customer.cust-num customer.name customer.address customer.city
    WITH FRAME cust-frame SCROLL 1 5 DOWN ATTR-SPACE.

FIND FIRST customer.
REPEAT counter = 1 TO 5:
    DISPLAY cust-num name address city WITH FRAME cust-frame.
    DOWN WITH FRAME cust-frame.
    FIND NEXT customer NO-ERROR.
    IF NOT AVAILABLE customer
    THEN LEAVE.
END.
UP 5 WITH FRAME cust-frame.
oldchoice = "".

REPEAT:
    STATUS DEFAULT "Enter C to create, D to delete".
    CHOOSE ROW customer.cust-num NO-ERROR GO-ON(CURSOR-RIGHT)
       WITH FRAME cust-frame.

    /* After choice */
    IF FRAME-VALUE = ""
    THEN NEXT.
    /* Force user to press END or move cursor to valid line */
    IF FRAME-VALUE <> oldchoice
    THEN DO:
        oldchoice = FRAME-VALUE.
        FIND customer WHERE cust-num = INTEGER(FRAME-VALUE).
    END.
```

r-chose1.p *(2 of 3)*

```
 /* React to moving cursor off the screen */
IF LASTKEY = KEYCODE("CURSOR-DOWN")
THEN DO:
     FIND NEXT customer NO-ERROR.
     IF NOT AVAILABLE customer
     THEN FIND FIRST customer.
     DOWN WITH FRAME cust-frame.
     DISPLAY cust-num name address city WITH FRAME cust-frame.
     NEXT.
END.
IF LASTKEY = KEYCODE("CURSOR-UP")
THEN DO:
    FIND PREV customer NO-ERROR.
    IF NOT AVAILABLE customer
    THEN FIND LAST customer.
    UP WITH FRAME cust-frame.
    DISPLAY cust-num name address city WITH FRAME cust-frame.
    NEXT.
END.
/* CHOOSE selected a valid key.  Check which key. */
IF LASTKEY = KEYCODE("c")
THEN DO: /* Open a space in the frame. */
    SCROLL FROM-CURRENT DOWN WITH FRAME cust-frame.
    CREATE customer.
    UPDATE cust-num name address city WITH FRAME cust-frame.
    oldchoice = INPUT cust-num.
    NEXT.
END.
IF LASTKEY = KEYCODE("d")
THEN DO: /* Delete a customer from the database. */
     DELETE customer.
     FIND NEXT customer NO-ERROR.
     /* Move to correct position in database. */
    IF NOT AVAILABLE customer
    THEN DO:
        FIND FIRST customer NO-ERROR.
        IF NOT AVAILABLE customer
        THEN DO:
            CLEAR FRAME cust-frame.
            UP WITH FRAME cust-frame.
            NEXT.
        END.
    END.
```

r-chose1.p *(3 of 3)*

```
      IF FRAME-LINE(cust-frame) = FRAME-DOWN(cust-frame)
      THEN DO:/* If last screen line deleted */
           DISPLAY cust-num name address city WITH FRAME cust-frame.
           NEXT.
      END.
      SCROLL FROM-CURRENT WITH FRAME cust-frame.
      REPEAT counter = 1 TO 100
            WHILE FRAME-LINE(cust-frame) < FRAME-DOWN(cust-frame):
          FIND NEXT customer NO-ERROR.
          IF NOT AVAILABLE customer
          THEN DO:
              FIND FIRST customer NO-ERROR.
             IF NOT AVAILABLE customer
             THEN LEAVE.
          END.
          DOWN WITH FRAME cust-frame.
          IF INPUT cust-num = ""
          THEN DISPLAY cust-num name address city WITH FRAME cust-frame.
      END.
      UP counter - 1 WITH FRAME cust-frame.
      oldchoice = INPUT cust-num.
   END.
END.
STATUS DEFAULT.
```

The SCROLL statement controls the scrolling action in the frame when you create and delete customers. To add a customer to the database, type **C**. Create opens a line in the frame and the SCROLL statement moves data below the line down. Then you type the new customer information into the frame. Type **D** to delete a customer from the database. When you delete a customer, all rows below the deleted customer row move up one row.

You can perform the same function with fewer statements if you do not use the SCROLL statement. You can substitute the `r-chose1.p` procedure segment with the `r-chose2.p` to perform the delete function.

r-chose2.p

```
                    .
                    .
                    .
IF LASTKEY = KEYCODE("d")
THEN DO: /* Delete a customer from the database. */
  DELETE customer.
  REPEAT counter = 1 TO 100 WHILE FRAME-LINE(cust-frame)
      <= FRAME-DOWN(cust-frame).
    FIND NEXT customer NO-ERROR.
    IF AVAILABLE customer
    THEN DISPLAY cust-num name address city
       WITH FRAME cust-frame.
    ELSE CLEAR FRAME cust-frame.
    DOWN WITH FRAME cust-frame.
  END.
  UP counter - 1 WITH FRAME cust-frame.
  oldchoice = INPUT cust-num.
END.
                    .
                    .
                    .
```

You can see the entire `r-chose2.p` procedure on-line. This example only shows the portion that is different from the `r-chose1.p` procedure.

The `r-cuhelp.p` procedure provides help for the cust-num field when a user presses **HELP**. It displays five customer names and numbers. The user can press (**UP-ARROW**), (**DOWN-ARROW**), to scroll down, or (**RETURN**) to exit.

r-cuhelp.p

```
FORM customer.cust-num customer.name
  WITH FRAME cust-frame 5 DOWN ROW 10 CENTERED
  OVERLAY TITLE " Available Customers ".

REPEAT WHILE FRAME-LINE(cust-frame) <= FRAME-DOWN(cust-frame):
  FIND NEXT customer.
  DISPLAY customer.cust-num customer.name
    WITH FRAME cust-frame.
    DOWN WITH FRAME cust-frame.
END.

UP 5 WITH FRAME cust-frame.

REPEAT:
  CHOOSE ROW customer.cust-num NO-ERROR
    WITH FRAME cust-frame.
  FIND customer WHERE customer.cust-num = INPUT
    customer.cust-num.
  IF KEYFUNCTION(LASTKEY) = "CURSOR-UP" THEN DO:
    FIND PREV customer NO-ERROR.
    IF AVAILABLE customer THEN DO:
      SCROLL DOWN WITH FRAME cust-frame.
      DISPLAY customer.cust-num customer.name
      WITH FRAME cust-frame.
    END.
  END.
  ELSE
  IF KEYFUNCTION(LASTKEY) = "CURSOR-DOWN" THEN DO:
    FIND NEXT customer NO-ERROR.
    IF AVAILABLE customer THEN DO:
      SCROLL UP WITH FRAME cust-frame.
      DISPLAY customer.cust-num customer.name
        WITH FRAME cust-frame.
    END.
  END.
  ELSE
  IF KEYFUNCTION(LASTKEY) = "RETURN" THEN DO:
    FRAME-VALUE = FRAME-VALUE.
    HIDE FRAME cust-frame.
    RETURN.
  END.
END.
```

See also CHOOSE statement, Frame phrase

SDBNAME function

Accepts an integer expression or a character expression as a parameter. If the parameter resolves to a currently connected non-OpenEdge database then the SDBNAME function returns the logical name of the schema holder database containing the non-OpenEdge schema. If the parameter resolves to a currently connected OpenEdge database, the SDBNAME function returns the logical name of this database.

Syntax

```
SDBNAME  ( { integer-expression | logical-name | alias } )
```

integer-expression

If the parameter supplied to SDBNAME is an integer expression, and there are, for example, three connected databases, then SDBNAME(1), SDBNAME(2), and SDBNAME(3) return the logical names of their respective schema holder databases. Also, if there are three connected databases, SDBNAME(4), SDBNAME(5), etc., return the Unknown value (?).

logical-name or *alias*

These forms of the SDBNAME function require a quoted character string or a character expression as a parameter. If the parameter is the logical name of a connected database or an alias of a connected database, then the logical name of the schema holder database is returned according to the rule. Otherwise, SDBNAME returns the Unknown value (?).

Example

This procedure displays schema holder databases, if applicable, for all connected databases:

r-sdbnm.p

```
DEFINE VARIABLE i AS INTEGER.
REPEAT i= 1 TO NUM-DBS:
  DISPLAY SDBNAME(i)
  SDBNAME(i) = LDBNAME(i)
  FORMAT "SCHEMA-HOLDER/SUB-SCHEMA        "
    COLUMN-LABEL "  DataServer!Classification".
END.
```

See also

ALIAS function, CONNECT statement, CONNECTED function,CREATE ALIAS statement, CREATE CALL statement, DATASERVERS function, DBCODEPAGE function, DBCOLLATION function, DBRESTRICTIONS function, DBTYPE function, DBVERSION function, DELETE ALIAS statement, DISCONNECT statement, FRAME-DB function, LDBNAME function, NUM-DBS function, PDBNAME function

SEARCH function

Searches the directories and libraries defined in the PROPATH environment variable for a file. The SEARCH function returns the full pathname of the file unless it is found in your current working directory. If SEARCH does not find the file, it returns the Unknown value (?).

Syntax

```
SEARCH ( opsys-file )
```

opsys-file

A character expression whose value is the name of the file you want to find. The name can include a complete or partial directory path. If *opsys-file* is a constant string, you must enclose it in quotation marks (" "). The value of *opsys-file* must be no more than 255 characters long.

Example

In this procedure, the SEARCH function returns the fully qualified pathname of the filename entered if it is not in the current working directory. If SEARCH cannot find the file, it returns the Unknown value (?). The procedure displays the fully qualified pathname or a message indicating that the file could not be found.

r-search.p

```
DEFINE VARIABLE fullname AS CHARACTER FORMAT "x(55)".
DEFINE VARIABLE filename AS CHARACTER FORMAT "x(20)".

REPEAT:
  UPDATE filename HELP "Try entering 'help.r' or 'dict.r'"
    WITH FRAME a SIDE-LABELS CENTERED.
    fullname = SEARCH(filename).
    IF fullname = ?
    THEN
      DISPLAY "UNABLE TO FIND FILE " filename
        WITH FRAME b ROW 6 CENTERED NO-LABELS.
    ELSE
      DISPLAY "Fully Qualified Path Name Of:" filename
      SKIP(2) "is:"
      fullname WITH FRAME c ROW 6 NO-LABELS CENTERED.
END.
```

Notes

- The SEARCH function is double-byte enabled. You can specify a filename with the *opsys-file* argument that contains double-byte characters.
- Use the SEARCH function to ensure that procedures that get input from external data files are independent of specific directory paths. The files must be in one of the directories or libraries defined in the PROPATH environment variable.
- Typically, the PROPATH includes a nil entry representing the current working directory. If the SEARCH function finds the file when searching this entry, it returns only the simple name of the file rather than the full pathname. If the PROPATH does not include a nil entry or another entry that specifies the current working directory, the SEARCH function does not search the current working directory.
- If you provide a fully qualified pathname, SEARCH checks if the file exists. In this case, SEARCH does not search directories on the PROPATH.
- When you search for a file that is in a library, SEARCH returns the file's pathname in the form *path-name*<<*member-name*>>, where *path-name* is the pathname of the library and *member-name* is the name of the file. The double angle brackets indicate that the file is a member of a library. For example, in the path /usr/apps.pl<<proc1.r>>, proc1.r is the name of the file in the library apps.pl.

  The LIBRARY function and MEMBER function use the special syntax to return, respectively, the library name and *member-name* of the file in the library.
- If an application repeatedly runs a procedure, you can improve performance by using the SEARCH function once to build a full pathname for that procedure. Use this value in the RUN statement to avoid repeated searches of the PROPATH.

- In Windows, you can specify URL pathnames on the PROPATH. If the file is found in a directory specified by a URL, SEARCH returns the full URL pathname of the file which includes the filename appended to the URL PROPATH entry. If you provide a fully-qualified URL, SEARCH checks if the file exists. In this case, SEARCH does not search URLs on the PROPATH. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the SEARCH function continues searching with the next PROPATH entry.

- If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-ICON( ), LOAD-SMALL-ICON( ), LOAD-IMAGE( ), LOAD-IMAGE-DOWN( ), LOAD-IMAGE-UP( ), LOAD-IMAGE-INSENSITIVE( ), or LOAD-MOUSE-POINTER( ) methods with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value with the load methods to avoid repeated searches of the PROPATH.

SEEK function

Returns the offset of the file pointer in a text file. You define a procedure variable to hold the offset value and later position the file to that offset.

Syntax

```
SEEK ( { INPUT | OUTPUT | name } )
```

INPUT

If you specify INPUT, the SEEK function returns the current position of the file pointer in the unnamed input stream.

OUTPUT

If you specify OUTPUT, the SEEK function returns the current position of the file pointer in the unnamed output stream.

name

If you specify SEEK (*name*), the SEEK function returns the current position of the file pointer in the named input or output stream. The stream must be associated with an open file, or SEEK returns the Unknown value (?).

Example

This procedure shows how you can use the SEEK function to access data in an text file. Using SEEK this way allows you to index into a non-indexed file.

r-seek1.p

```
DEFINE VARIABLE itemno LIKE item.item-num.
DEFINE VARIABLE itdesc LIKE item-name.
DEFINE VARIABLE m-pos AS INT.

SET itemno LABEL
  "Select a record number to position the output file" WITH SIDE-LABELS.
OUTPUT TO test.fil.
FIND item WHERE itemno = item-num.

IF item-num = itemno
THEN m-pos = SEEK(OUTPUT).

EXPORT item-num item-name.
OUTPUT CLOSE.

INPUT FROM test.fil.
SEEK INPUT TO m-pos.
SET itemno itdesc WITH FRAME d2.
INPUT CLOSE.
```

In the example, you are prompted to select an item number to position the output file. When a record is found with that item number, the SEEK function returns the offset into the variable m-pos. The value for m-pos is the current value of the file pointer. The SEEK statement uses the value in m-pos to position the file pointer in the unnamed input stream.

Notes

- The first byte in a file is byte 0.
- You cannot use the SEEK function with the INPUT THROUGH statement, the INPUT-OUTPUT THROUGH statement, or the OUTPUT THROUGH statement. When used with one of these statements, the SEEK function returns the Unknown value (?).
- After you assign the value of the SEEK function to a procedure variable, you can use that value to reposition the file in the event of an error.
- For more information on streams, see the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces*.

See also

DEFINE STREAM statement, INPUT FROM statement, OUTPUT TO statement, SEEK statement

SEEK statement

Positions the file pointer to a user-defined offset in a text file. This statement does not require you to close and reopen the file.

Syntax

```
SEEK { INPUT | OUTPUT | STREAM stream }
  TO { expression | END }
```

INPUT

If you specify INPUT, the SEEK statement positions the file pointer in the unnamed input stream.

OUTPUT

If you specify OUTPUT, the SEEK statement positions the file pointer in the unnamed output stream.

STREAM *stream*

If you specify STREAM *stream*, the SEEK statement positions the file pointer in the named input or output stream. If you do not name a stream, Progress uses the unnamed stream.

TO *expression*

An expression whose value is an integer that indicates the byte location to position the file pointer. If *expression* equals 0, the file pointer is positioned to the first byte in the file. If you want to position the pointer to the last byte in the file, but you do not know the offset, use END.

END

Positions the pointer to the last byte in the file.

Example

Since text file formats differ on each machine, the SEEK function does not necessarily return a number that is meaningful to anyone, but it is meaningful to the SEEK statement. With the exception of SEEK to 0 or SEEK TO END, any address used in the SEEK statement is only guaranteed to behave consistently if the address was previously derived from the SEEK function. Therefore, an expression such as SEEK TO SEEK (INPUT) -n might work differently on different operating systems. Record delimiters must be new-lines on UNIX, and carriage-return/linefeed pairs on all others.

r-seek.p

```
/* This procedure seeks to the end-of-file, collects the seek address,
and writes a record. The record is subsequently retrieved using the SEEK
statement on the stashed seek address. */

DEFINE VAR savepos AS INT.
DEFINE VAR c       AS CHAR FORMAT "x(20)".
OUTPUT TO seek.out APPEND NO-ECHO.
savepos = SEEK(OUTPUT).
PUT UNFORMATTED "abcdefg" SKIP.
OUTPUT CLOSE.
INPUT FROM seek.out NO-ECHO.
SEEK INPUT TO savepos.
SET c.
DISPLAY c.
INPUT CLOSE.
```

Notes

- The SEEK statement does not work with named streams identified in the INPUT-THROUGH, OUTPUT-THROUGH, or INPUT-OUTPUT-THROUGH statements.
- An expression such as SEEK TO SEEK (INPUT) -n might work differently on different operating systems.
- For more information on streams, see the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces*.

See also

DEFINE STREAM statement, INPUT FROM statement, OUTPUT TO statement, SEEK function

SELECTION-LIST phrase

Describes the selection-list representation of a field or variable. A selection-list is a scrollable list of values. The SELECTION-LIST phrase is an option of the VIEW-AS phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
SELECTION-LIST
  [ SINGLE | MULTIPLE ]
  [ NO-DRAG ]
  { LIST-ITEMS item-list | LIST-ITEM-PAIRS item-pair-list }
  [ SCROLLBAR-HORIZONTAL ]
  [ SCROLLBAR-VERTICAL ]
  {   size-phrase
    | { INNER-CHARS cols INNER-LINES rows }
  }
  [ SORT ]
  [ TOOLTIP tooltip ]
```

SINGLE

Specifies that on input the user can select only a single item from the list. This is the default. The value of the selection-list is set to the character-string item the user selects.

MULTIPLE

Specifies that on input the user can select one or more items from the item list. The value of the selection-list is set to a comma-separated list of character-string items that the user selects.

NO-DRAG

Specifies that the user cannot select items by simultaneously holding down the mouse select button and dragging the mouse through the list.

If you specify NO-DRAG then the DRAG-ENABLED attribute is set to FALSE. You can set the DRAG-ENABLED attribute only before the selection-list is realized. The default is TRUE.

In Windows, DRAG-ENABLED is always TRUE and the NO-DRAG option is ignored.

LIST-ITEMS *item-list*

Specifies the items to appear in the list. *item-list* represents a comma-separated list of character-string constants.

LIST-ITEM-PAIRS *item-pair-list*

Specifies a list of label-value pairs. Each pair represents a label and value of the associated field or variable. When the selection-list appears, it displays each pair's label. Then, if the user selects a label, Progress assigns the corresponding value to the field or variable. The syntax for *item-pair-list* is as follows:

```
label , value [ , label , value ] ...
```

label

A character string representing the label of the field or variable.

value

A valid value for the field or variable.

SCROLLBAR-VERTICAL

Specifies that a scroll bar is displayed along side the selection-list. The user can browse through a long selection-list by manipulating the slider.

SCROLLBAR-HORIZONTAL

Specifies that a scroll bar is displayed along the bottom of the selection-list. The user can view long list items by manipulating the slider.

size-phrase

Specifies the outside dimensions of the selection-list widget. Following is the syntax for the *size-phrase*:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information, see the SIZE phrase reference entry.

INNER-CHARS *cols* INNER-LINES *rows*

Specifies the number of character positions visible in each line of the selection-list and the number of lines visible in the selection-list. Both *cols* and *rows* must be integer constants.

Note that the values you supply for INNER-CHARS and INNER-LINES specify only the size of the list, not the overall size of the selection-list widget. The overall size is determined by the size of the list plus the sizes of the margin and border heights and widths.

SORT

Specifies that list items are sorted prior to display.

TOOLTIP *tooltip*

Allows you to define a help text message for a text field or text variable. Progress automatically displays this text when the user pauses the mouse button over a text field or text variable for which a tooltip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the tooltip is removed. No tooltip is the default. The TOOLTIP option is supported in Windows only.

Example

The `r-select.p` procedure prompts the user for a directory name and then populates a selection-list with the contents of the specified directory. After the user selects an item from the selection-list, the procedure echoes back the selection.

The procedure uses the INPUT FROM statement to read the contents of the user-specified directory and creates a comma-separated list of all the file and directory names in the directory. It then assigns the comma-separated list to the LIST-ITEMS attribute of the selection-list. Because an assignment to an attribute depends on the widget being located in a frame, the DEFINE FRAME statement is used to locate the selection-list.

r-select.p

```
DEFINE STREAM dirlist.
DEFINE VARIABLE f-name AS CHARACTER FORMAT "x(14)".
DEFINE VARIABLE choice AS CHARACTER FORMAT "x(50)"
                           LABEL "You have selected".
DEFINE VARIABLE list_contents AS CHARACTER FORMAT "x(200)" init "".
DEFINE VARIABLE dir AS CHARACTER FORMAT "x(40)" init ""
                           LABEL "Please enter a directory pathname ".
DEFINE VARIABLE sl AS CHARACTER VIEW-AS SELECTION-LIST
                           INNER-CHARS 15 INNER-LINES 10 SORT.
DEFINE FRAME b sl.
DEFINE FRAME c choice.

ENABLE dir WITH FRAME d WITH SIDE-LABELS.
ON RETURN OF dir IN FRAME d DO:
  ASSIGN FRAME d dir.
  INPUT STREAM dirlist FROM OS-DIR (dir).
  IMPORT STREAM dirlist f-name.
  list_contents = f-name.
  REPEAT:
    IMPORT STREAM dirlist f-name.
    list_contents = list_contents + "," + f-name.
  END.
  INPUT CLOSE.

  sl:LIST-ITEMS IN FRAME b = list_contents.
 ENABLE sl WITH FRAME b NO-LABELS TITLE "Please Select a File" WIDTH 50.
END.

ON VALUE-CHANGED OF sl IN FRAME b DO:
choice = sl:SCREEN-VALUE.
DISPLAY choice WITH FRAME c SIDE-LABELS.
END.
WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- When the selection-list appears, if it contains the value of the associated field or variable, that value is initially highlighted. Otherwise, no value in the selection-list is initially highlighted.
- The LIST-ITEMS option of the SELECTION-LIST phrase requires a list of quoted items ("a", "b", "c"), whereas the LIST-ITEMS attribute of a selection-list requires a quoted list of items ("a, b, c"). Similarly, the LIST-ITEM-PAIRS option of the SELECTION-LIST phrase requires a list of quoted items ("a", "1", "b", "2", "c", "3"); whereas the LIST-ITEM-PAIRS attribute of a selection-list requires a quoted list of items ("a, 1, b, 2, c, 3").
- If you specify the SORT option for a selection-list, then any items you add with ADD-FIRST, ADD-LAST, or INSERT methods are added in sorted order rather than the order you specify.
- In Windows, you can use a mnemonic to transfer focus to the selection-list.

See also SIZE phrase, VIEW-AS phrase

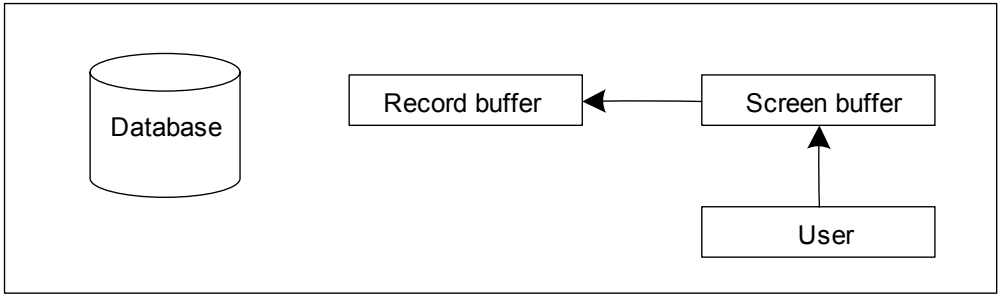
SET statement

Requests input, and then puts the data in the screen buffer frame and in the specified fields or variables. The SET statement is a combination of these statements:

- **PROMPT-FOR** — Prompts the user for data and puts that data into the screen buffer.
- **ASSIGN** — Moves data from the screen buffer to the record buffer.

Note: Does not apply to SpeedScript programming.

Data movement



Syntax

```
SET
[ STREAM stream ]
[ UNLESS-HIDDEN ]
[
  {  field [ view-as-phrase ] [ format-phrase ] }
       [ WHEN expression ]
    | TEXT ( field [ format-phrase ] ... )
    | field = expression
    | constant [ AT n | TO n ]
    | ^
    | SPACE [ ( n ) ]
    | SKIP [ ( n ) ]
] ...
[ GO-ON (key-label ... ) ]
[ frame-phrase ]
[ editing-phrase ]
[ NO-ERROR ]
```

```
SET [STREAM stream] [UNLESS-HIDDEN]
       record [EXCEPT field. . .] [frame-phrase]
    [NO-ERROR]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

UNLESS-HIDDEN

Restricts SET to fields whose HIDDEN attribute is FALSE.

field

Represents the name of the field or variable whose value you want to store in the screen buffer and in the field or variable.

In the case of array fields, array elements with constant subscripts are treated as any other field. Array fields with no subscripts are expanded as though you typed in the implicit elements. See the DISPLAY statement reference entry for information on how Progress handles array fields with expressions as subscripts.

view-as-phrase

Specifies the widget used to represent the field. For more information on *view-as-phrase*, see the VIEW-AS phrase reference entry.

format-phrase

Specifies one or more frame attributes for a field, variable, or expression. For more information on *format-phrase*, see the Format phrase reference entry.

WHEN *expression*

Sets the field only when *expression* has a value of TRUE. An *expression* is a field name, variable name, or expression whose value is logical.

TEXT

Defines a group of character fields or variables (including array elements) to use automatic word wrap. The TEXT option works with character fields only. When you insert data in the middle of a TEXT field, Progress wraps data that follows into the next TEXT field, if necessary. If you delete data from the middle of a TEXT field, Progress wraps data that follows into the empty area. If you enter more characters than the format for the field allows, Progress discards the extra characters. The character fields must be in the x(*n*) format.

A blank in the first column of a line marks the beginning of a paragraph. Lines within a paragraph are treated as a group and will not wrap into other paragraphs.

Table 46 lists the keys you can use within a TEXT field, and their actions.

Table 46: Key actions in a TEXT field

Key	Action
APPEND-LINE	Combines the line the cursor is in with the next line.
BACK-TAB	Moves the cursor to the previous TEXT field.
BREAK-LINE	Breaks the current line into two lines beginning with the character the cursor is on.
BACKSPACE	Moves the cursor one position to the left and deletes the character at that position. If the cursor is at the beginning of a line, BACKSPACE moves the cursor to the end of the previous line.
CLEAR	Clears the current field and all fields in the TEXT group that follow.
DELETE-LINE	Deletes the line the cursor is in.
NEW-LINE	Inserts a blank line below the line the cursor is in.
RECALL	Clears fields in the TEXT group and returns initial data values for the group.
RETURN	In overstrike mode, moves to the next field in the TEXT group on the screen. In insert mode, the line breaks at the cursor and the cursor is positioned at the beginning of the new line.
TAB	Moves to the field after the TEXT group on the screen. If there is no other field, the cursor moves to the beginning of the TEXT group.

In this procedure, the s-com, or Order Comments field is a TEXT field. Run the procedure and enter text in the field to see how the TEXT option works:

r-text.p

```
DEFINE VARIABLE s-com AS CHARACTER FORMAT "x(40)" EXTENT 5.

FORM "Shipped   :" order.ship-date AT 13 SKIP
     "Misc Info :" order.instructions AT 13 SKIP(1)
     "Order Comments :" s-com AT 1
WITH FRAME o-com CENTERED NO-LABELS TITLE "Shipping Information".

FOR EACH customer, EACH order OF customer:
    DISPLAY cust.cust-num cust.name order.order-num order.order-date
            order.promise-date WITH FRAME order-hdr CENTERED.
    UPDATE ship-date instructions TEXT(s-com) WITH FRAME o-com.
    s-com = "".
END.
```

field = expression

Indicates that the value of *field* is determined by evaluating the *expression* rather than having it entered on the screen or from a file. An assignment statement is embedded within the SET statement.

constant AT *n*

A constant value that you want to display in the frame. The *n* is the column in which you want to start the display.

constant TO *n*

A constant value that you want to display in the frame. The *n* is the column in which you want to end the display.

^

Tells Progress to ignore an input field when input is being read from a file. Also, the following statement reads a line from an input file and ignores that line.

```
SET ^
```

This is an efficient way to skip over lines.

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces to insert after the expression on the display. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, Progress starts a new line and discards any extra spaces. If you do not use this option or do not use *n*, Progress inserts one space between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines to be inserted after the expression is displayed. The *n* can be 0. If you do not use this option, a line is not skipped between expressions only if they do not fit on one line. If you use the SKIP option, but do not specify *n*, or if *n* is 0, a new line is started unless it is already at the beginning of a new line.

GO-ON (*keylabel* . . .)

Tells Progress to take the GO action when the user presses any of the keys listed. The keys you list are used in addition to keys that perform the GO action by default or because of ON statements. When you list a key label in the GO-ON option, you use the keyboard label of that key. For example, if you want Progress to take the GO action when the user presses **F1**, you use the statement GO-ON(**F1**). If you list more than one key, separate them with spaces, not commas, as in GO-ON(**F1 RETURN**).

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

editing-phrase

Supported only for backward compatibility.

Identifies processing to take place as each keystroke is entered. This is the syntax for the *editing-phrase*:

```
[ label: ] EDITING: statement ... END.
```

For more information on *editing-phrase*, see the EDITING phrase reference entry.

record

Represents the name of a record buffer. All of the fields in the record, except those with the data type RECID and ROWID, are processed exactly as if you set each individually. The record you name must contain at least one field.

To use SET with a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

NO-ERROR

Specifies that any errors that occur in the attempt to set the value are suppressed. After the SET statement completes, you can check the ERROR-STATUS system handle for information on any errors that occurred.

EXCEPT *field*

Affects all fields except those fields listed in the EXCEPT phrase.

Examples

The `r-set.p` procedure reads each item record, displays the item-num and lets the user enter information for the item-name, on-hand, allocated, and price fields. When you run this procedure, notice that it does not display existing values for the item-name, on-hand, allocated, and price fields.

r-set.p

```
FOR EACH item:
  DISPLAY item-num.
  SET item-name on-hand allocated price.
END.
```

On each iteration of the block, the FOR EACH statement reads a single record into the item record buffer. The DISPLAY statement moves the item-num from the record buffer to the screen buffer where you can see it. The SET statement prompts for data, stores the data in screen buffers, and moves the data to the record buffer, overwriting whatever is already there. Therefore, even though the item-name, on-hand, allocated, and price fields are put into the item record buffer by the FOR EACH statement, you never see the values for those fields.

The r-set2.p procedure displays the cust-num, name, and credit-limit for a customer and lets you change the credit-limit field. The HELP option in the SET statement displays help information at the bottom of the screen when you are changing the credit-limit. The VALIDATE option in the SET statement makes sure that the credit-limit value is greater than 0. If it isn't, VALIDATE displays the message "Invalid credit limit."

r-set2.p

```
FOR EACH customer:
  DISPLAY cust-num name credit-limit.
  SET name credit-limit
    VALIDATE(credit-limit > 0, "Invalid credit limit.")
    HELP "Enter a positive credit-limit.".
  REPEAT:
    CREATE order.
    cust-num = customer.cust-num.
    SET order-num
      ship-date VALIDATE(ship-date > today,
      "Ship date too early.").
  END.
END.
```

After you modify credit-limit, the procedure creates an order for the customer and assigns the customer.cust-num value to the cust-num field in the order record. The SET statement lets you enter information for the order-num and ship-date fields. The VALIDATE option in the SET statement makes sure that the ship date is greater than TODAY.

Notes

- If any *field* is a field in a database record, the SET statement upgrades the record lock condition to EXCLUSIVE-LOCK before updating the record.
- If any *field* is part of a record retrieved with a field list, the SET statement rereads the complete record before updating it.
- SET does not move data into the field or variable if there is no data in the corresponding screen field. There is data in a screen field if a DISPLAY of the field was done or if you enter data into the field. If you set a field or variable that has not been DISPLAYed in the frame and key in blanks, then the field or variable is not changed because the screen field is changed only if the data differs from what was in the frame field.
- When Progress compiles a procedure, it designs all the frames used by that procedure. When you run the procedure, the SET statement puts data into those fields.

- In a SET statement, Progress first prompts for all specified fields and then assigns the values of those fields, moving from left to right. During this left to right pass of the field list, Progress processes embedded assignments (*field = assignment*) as it encounters them.
- If you are getting input from a device other than the terminal, and the number of characters read by the SET statement for a particular field or variable exceeds the display format for that field or variable, Progress returns an error. However, if you are setting a logical field that has a format of y/n and the data file contains a value of YES or NO, Progress converts that value to “y” or “n”.
- If you type blanks into a field in which data has never been displayed, the ENTERED function returns FALSE and the SET or ASSIGN statement does not update the underlying field or variable. Also, if Progress has marked a field as entered, and the SET statement prompts for the field again and you do not enter any data, Progress no longer considers the field an entered field.
- If you use a single qualified identifier with the SET statement, the Compiler first interprets the reference as *dbname.tablename*. If the Compiler cannot resolve the reference as *dbname.tablename*, it tries to resolve it as *tablename.fieldname*.

  When using SET to set fields, you must use table names that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.
- The SET statement causes the ASSIGN and WRITE events to occur and fires all related database ASSIGN and WRITE triggers. The ASSIGN triggers execute before the WRITE triggers and after the field is actually updated. The WRITE triggers only execute if the ASSIGN triggers do not return an error. If an ASSIGN trigger fails (or executes a RETURN statement with the ERROR option), the SET statement is undone. This means that any changes to the database from that SET statement are backed out. If the SET statement occurs within a transaction, any changes to variables, worktable fields, and temporary table fields are also undone unless they are defined with the NO-UNDO option. Also, if a WRITE trigger fails (or executes a RETURN statement with the ERROR option), the SET statement is undone. See *OpenEdge Development: Progress 4GL Handbook* for more information on database triggers.

See also DEFINE STREAM statement, EDITING phrase, Format phrase, Frame phrase, PROMPT-FOR statement, UPDATE statement

SET-BYTE-ORDER statement

Sets an internal indicator designating the byte-order of the data pointed to by the MEMPTR variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
SET-BYTE-ORDER( memptr ) = integer-expression
```

memptr

An expression that returns a MEMPTR.

integer-expression

An expression that returns an integer value that will be used to indicate the byte-ordering of the data in the memory to which the MEMPTR points. *integer-expression* must be one of the reserved keywords defined in Table 47 or its corresponding value. If *integer-expression* is not valid, Progress generates an error.

Table 47: Byte order options

Keyword	Value	Description
HOST-BYTE-ORDER	1	Same format as the machine where the process that calls SET-BYTE-ORDER is running.
BIG-ENDIAN	2	A multiple-byte data type is stored with the high-order byte in the lowest address reserved for the data; successively lower-order bytes are stored at successively higher addresses. Note that Internet protocols use BIG-ENDIAN byte-ordering.
LITTLE-ENDIAN	3	A multiple-byte data type is stored with the low-order byte in the lowest address reserved for the data; successively higher-order bytes are stored at successively higher addresses.

Notes

- The byte order for a MEMPTR is HOST-BYTE-ORDER by default, that is, if SET-BYTE-ORDER has not been called for a given MEMPTR, its byte order is HOST-BYTE-ORDER.
- SET-BYTE-ORDER by itself never affects data currently in the MEMPTR, that is, it does not actually re-order the data. It only affects how subsequent calls to the PUT- statements and GET- functions work with that MEMPTR variable.

See also GET-BYTE-ORDER function

SET-DB-CLIENT function

Uses the user ID represented by a sealed and validated Client-principal object to set a client user ID for the specified OpenEdge database. Returns TRUE if the user ID is set successfully; otherwise, it returns FALSE.

Note: Using this function overrides the database user ID previously set by either the SET-CLIENT( ) method or the SETUSERID function.

When a user ID is set on a connected database, Progress uses that user ID to determine whether the user has permission to access tables and fields in that particular database.

Syntax

```
SET-DB-CLIENT( client-principal-handle
  [, integer-expression | logical-name | alias ] )
```

client-principal-handle

A handle to a sealed Client-principal object. The Client-principal object must be created in an authentication domain that is registered in the database connection registry. If the Client-principal object is not sealed, or the handle is the Unknown value (?), Progress generates a run-time error and the current user ID remains unchanged.

integer-expression

The sequence number of a connected database for which to set the user ID. For example, SET-DB-CLIENT(hcp, 1) sets the user ID associated with the specified Client-principal object for the first database, SET-DB-CLIENT(hcp, 2) sets the user ID for the second database, and so on. If you specify a sequence number that does not correspond to a connected database, Progress generates a run-time error.

logical-name or *alias*

The logical name or alias of a connected database for which to set the user ID. These forms require a quoted character string or a character expression. If you specify a logical name or alias that does not correspond to a connected database, Progress generates a run-time error.

If you do not specify a database, Progress sets the client user ID for all connected databases (which overrides the user ID previously set by a call to the SET-CLIENT( ) method).

If the LOGIN-STATE attribute for the sealed Client-principal object is not "LOGIN", Progress generates a run-time error and the current user ID remains unchanged.

Progress also validates the Client-principal object using the database connection's domain registry. If not valid, Progress generates a run-time error and the current user ID remains unchanged.

You can also use this function, instead of the SETUSERID function, to set a database user ID when the user ID is not in the _User table.

Calling this method generates an audit event, and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

See also Client-principal object handle, SET-CLIENT( ) method, SETUSERID function

SET-POINTER-VALUE statement

Sets a variable of type MEMPTR to the value of a particular memory location.

Note: Does not apply to SpeedScript programming.

Syntax

```
SET-POINTER-VALUE ( memptr-value ) = memptr-expression
```

memptr-value

A 32-bit integer that represents a memory location.

memptr-expression

An expression that evaluates to a value of type MEMPTR.

Example

The following example calls a DLL routine that returns a pointer to a structure, extracts an address at byte 5 of the structure, uses SET-POINTER-VALUE to assign the address to a Progress MEMPTR, and displays the character string at the address:

```
DEFINE VARIABLE person_struct AS MEMPTR. /* pointer to structure */
DEFINE VARIABLE name AS MEMPTR. /* pointer to name */
SET-SIZE(person_struct) = 8.

PROCEDURE person_info EXTERNAL "person.dll" PERSISTENT:
  DEFINE OUTPUT PARAMETER person_struct AS MEMPTR.
END PROCEDURE.

RUN person_info(OUTPUT person_struct).
SET-POINTER-VALUE(name) = GET-LONG(person_struct,5).
DISPLAY GET-STRING(name,1) FORMAT "x(50)".
```

Notes

- SET-POINTER-VALUE is particularly useful when accessing Windows Dynamic Link Library (DLLs) or UNIX shared library routines from the 4GL. For more information on DLLs, see the chapter on DLLs in *OpenEdge Development: Programming Interfaces*.
- For more information on the MEMPTR data type, see *OpenEdge Development: Programming Interfaces*.

See also

GET-POINTER-VALUE function, SET-SIZE statement

SET-SIZE statement

Manages memory associated with a MEMPTR variable. This includes allocating and associating a region of memory with an uninitialized MEMPTR variable, setting the size of a region allocated with a Windows dynamic link library (DLL) or UNIX shared library routine for a MEMPTR, and deallocating memory associated with a MEMPTR variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
SET-SIZE ( memptr-var ) = size
```

memptr-var

A reference to a variable defined as MEMPTR.

size

An integer expression that specifies the allocated byte size of the region pointed to by *memptr-var*.

Example

In the following example, the SET-SIZE statement allocates 8 bytes of memory, associates the memory with the `ElipRegion` variable, and then initializes the region with four SHORT (2-byte) values:

r-setsiz.p

```
DEFINE VARIABLE ElipRegion AS MEMPTR.

SET-SIZE(ElipRegion) = 8.
PUT-SHORT(ElipRegion, 1) = 10.
PUT-SHORT(ElipRegion, 3) = 10.
PUT-SHORT(ElipRegion, 5) = 200.
PUT-SHORT(ElipRegion, 7) = 50.
```

Notes

- If *memptr-var* has no memory allocated to it (is uninitialized), then the SET-SIZE statement allocates a memory region of the specified size.
- If a MEMPTR variable is returned from a DLL or UNIX shared library routine that also allocates a memory region to it, then the SET-SIZE statement initializes the size of the existing region. Progress does not allocate a new region. This allows Progress to perform bounds checking on references to MEMPTR regions allocated outside Progress.

> **Caution:** You must know and specify the exact size of the memory region returned by the DLL routine from the type of structure it allocates. An incorrect size can result in data loss.

- If the specified size is 0, the SET-SIZE statement deallocates (frees) any memory associated with *memptr-var*, making it available to reference a new memory region.
- If the specified size is greater than 0 and *memptr-var* is fully initialized (associated with a memory region of a specified size), the SET-SIZE statement has no effect and leaves *memptr-var* unchanged.
- After initializing a MEMPTR variable, you can obtain the address of (or pointer to) the region associated with the variable using the GET-POINTER-VALUE function. Use this to build structures that contain pointers to other structures, as required by some DLL or UNIX shared library routines.
- For more information on accessing DLL routines from Progress, see *OpenEdge Development: Programming Interfaces*.

See also GET-POINTER-VALUE function, GET-SIZE function

SETUSERID function

Returns a TRUE value and assigns the user ID to the user if the user ID and password supplied to the SETUSERID function are in the _User table. If the user ID is not in the _User table or the password is incorrect, SETUSERID returns a FALSE value and does not assign the user ID to the user.

Note: Using this function overrides the database user ID previously set by either the SET-CLIENT( ) method or the SET-DB-CLIENT function.

Syntax

```
SETUSERID ( userid , password [ , logical-dbname ] )
```

userid

A constant, field name, variable name, or expression that results in a character value that represents the user's user ID. If you use a constant, you must enclose it in quotation marks (" ").

password

A constant, field name, variable name, or expression that results in a character value that represents the user's password. If you use a constant, you must enclose it in quotation marks.

logical-dbname

The logical name of the database where you want to check and set your user ID. The logical database name must be a character string enclosed in quotes, or a character expression. If you do not specify this argument, the Compiler inserts the name of the database that is connected when the procedure is compiled. If you omit this argument and more than one database is connected, it results in a Compiler error.

Example

To use the `login.p` procedure that is provided with Progress, you must define user IDs and passwords for those users who are authorized to access the database.

r-login1.p

```
/* Prompt user for userid and password and set the userid */

DEFINE VARIABLE id       LIKE _User._Userid.
DEFINE VARIABLE password LIKE _Password.
DEFINE VARIABLE tries    AS INTEGER NO-UNDO.

IF USERID("DICTDB") <> "" OR NOT CAN-FIND(FIRST DICTDB._User)
THEN RETURN.

DO ON ENDKEY UNDO, RETURN:  /* return if they hit endkey */
  /* Reset id and password to blank in case of retry */
  id = "".
  password = "".
  UPDATE SPACE(2) id SKIP  password BLANK
    WITH CENTERED ROW 8 SIDE-LABELS ATTR-SPACE
    TITLE " Database " + LDBNAME("DICTDB") + " ".
  IF NOT SETUSERID(id,password,"DICTDB") THEN DO:
    MESSAGE "Sorry, userid/password is incorrect.".
    IF tries > 1 THEN QUIT.   /* Only allow 3 tries */
    tries = tries + 1.
    UNDO, RETRY.
  END.
  HIDE ALL.
  RETURN.
END.
QUIT.
```

The `login.p` procedure uses the SETUSERID function to check the value of the user ID and password that a user enters. If the value of the function is FALSE, the procedure allows the user another try. The user has three tries to log in. The first time, the tries variable is 0; tries is 1 the second time, and 2 the third. The third time, tries is greater than 1 and the procedure exits from Progress with the QUIT statement.

Notes

- Use the Userid (-U) parameter together with the Password (-P) parameter. Progress checks the _User table for the userid supplied with the -U parameter. When it finds that userid, it compares the password supplied with the -P parameter with the password in the _User table. If the two passwords match, Progress assigns that userid to the OpenEdge session.
- Under the following conditions, the SETUSERID function returns a value of FALSE and does not assign a user ID to the user:
  - There are no entries in the _User table.
  - There is no _User record with the same user ID as the one supplied with the SETUSERID function.
  - The password supplied with the SETUSERID function does not match the password in the _User table record of the specified user ID.
- When using the SETUSERID function, Progress returns a compiler error under the following conditions:
  - There is no database connected.
  - The *logical-dbname* argument is omitted, and more than one database is currently connected.
- When specifying the *logical-dbname* argument, you must provide the name of the logical database, not the physical database.
- SETUSERID encodes the *password* argument and then compares the result with the value stored in the _User._password field of the _User table.
- After SETUSERID returns a value of TRUE and assigns a user ID to a user:
  - Progress uses that user ID when the user compiles procedures.
  - Subsequent uses of the USERID function return the assigned user ID.

- If the root user ID does not exist in the _User table, SETUSERID returns a value of FALSE when supplied with a userid of root. If the _User table does have a root entry, the user who assumes that user ID has all the privileges associated with the root user ID on UNIX.
- You must create a blank user ID ("") if you want to set the user ID to a null value.
- Table 48 shows how Progress determines a user ID on UNIX.

Table 48: Determining a UNIX user ID

Are there records in the _User table?	Are the -U and -P startup options supplied?	User ID
NO	YES	Error: -U and -P not allowed unless there are entries in the _User table.
NO	NO	UNIX user ID.
YES	NO	"" (blank user ID).
YES	YES	If the -U *userid* and -P *password* match those in the _User table, use that *userid*. Otherwise, do not assign a user ID.

- Table 49 shows how Progress determines a user ID in Windows.

Table 49: Determining a Windows user ID

Are there records in the _User table?	Are the -U and -P startup options supplied?	User ID
NO	YES	Error: -U and -P not allowed unless there are entries in the _User table.
NO	NO	"" (blank user ID) or operating system user ID if available.
YES	NO	"" (blank user ID).
YES	YES	If the -U *userid* and -P *password* match those in the _User table, use that user ID. Otherwise, do not assign a user ID.

- See *OpenEdge Development: Programming Interfaces* and *OpenEdge Data Management: Database Administration* for more information on user privileges.
- Calling this method generates an audit event, and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.
- Once an initial database connection is established, you can also use the SET-CLIENT( ) method (on the SECURITY-POLICY system handle) or the SET-DB-CLIENT function to set the user ID for the connection.

See also CONNECT statement, SET-CLIENT( ) method, SET-DB-CLIENT function, USERID function

SHA1-DIGEST function

Hashes the specified data using the United States Government Secure Hash Algorithm (SHA-1), and returns a 20-byte binary message digest value as a RAW value.

Syntax

```
SHA1-DIGEST( data-to-hash [ , hash-key ] )
```

data-to-hash

The source data to hash. The data may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the data is a CHARACTER or LONGCHAR value, Progress converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value.

hash-key

An optional key value to use in the hash operation. The key may be of type CHARACTER, LONGCHAR, RAW, or MEMPTR. If the key is a CHARACTER or LONGCHAR value, Progress converts it to UTF-8 (which ensures a consistent value regardless of code page settings). To avoid this automatic conversion, specify a RAW or MEMPTR value. This key value is combined with the source data before the hash operation begins.

If the *hash-key* value contains a null character, the null character is included in the hash operation.

See also

MD5-DIGEST function

SHOW-STATS statement

Writes procedure call statistics to the `proc.mon` output file if you specify the Statistics with Cross-Reference (-yx) parameter. It also writes procedure access and usage statistics to the `client.mon` output file if you specify the Statistics (-y) parameter, Statistics with **CTRL+C** (`-yc`) parameter, Segment Statistics (`-yd`) parameter, or Statistics with Cross-Reference (`-yx`) parameter. If you specify Segment Statistics (`-yd`), it also displays statistics for each code segment.

Ordinarily, when you specify these startup parameters, Progress writes the statistics to the output files at the end of your OpenEdge session. This might not be what you want. For example, if you start Progress using the `-y` or `-yc` parameters, you might want to view the execution buffer statistics as they occur during your OpenEdge session. With SHOW-STATS, you can force Progress to write the statistics at a specific time, instead of at session end. For more information on these startup parameters, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

The SHOW-STATS statement also writes the value of the STARTUP-PARAMETERS attribute to the `client.mon` output file.

Syntax

```
SHOW-STATS [ CLEAR ]
```

CLEAR

Resets all counters and timers that Progress uses to monitor the procedure call, procedure access, and usage statistics.

Example

This procedure runs the Data Dictionary and writes the procedure call, procedure access, and usage statistics to the `proc.mon` and `client.mon` output files:

r-stats.p

```
RUN dict.p.
SHOW-STATS.
```

Notes

- If you use the SHOW-STATS statement without specifying the Statistics (y) parameter, Progress opens the `client.mon` file as if you were dynamically specifying -y. However, the first SHOW-STATS statement that you use does not send any statistics to the `client.mon` file; it only opens the file. All subsequent SHOW-STATS statements, however, send procedure access and usage statistics to the file. But since you did not specify -y at startup, Progress does not write any startup parameter statistics to the `client.mon` file.
- You must specify the Statistics with Cross-Reference (-yx) parameter, if you want the SHOW-STATS statement to write procedure call statistics to the `proc.mon` file.

SIZE phrase

Specifies the width and height of a widget. You can express the dimensions in either character units or pixels.

Syntax

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

{ SIZE **|** SIZE-CHARS **}**

Specifies that the unit of measure is characters.

SIZE-PIXELS

Specifies that the unit of measure is pixels.

width

Specifies the width of the widget. If the units are characters, *width* must be a decimal constant. If the units are pixels, *width* must be an integer constant.

height

Specifies the height of the widget. If the units are characters, the value *height* must be a decimal constant. If the units are pixels, *height* must be an integer constant.

Example

The following example uses SIZE phrases to set the initial dimensions of the rectangle rec and to set the dimensions of the frame sz-frame. When you choose the b_size button, the rectangle is randomly resized.

r-size.p

```
DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE QUIT.
  END.

DEFINE BUTTON b_size LABEL "Size It".
DEFINE RECTANGLE rec SIZE 5 BY 5.
DEFINE FRAME butt-frame
  b_size b_quit
WITH CENTERED ROW SCREEN-LINES - 2.

DEFINE FRAME sz-frame
  SKIP(1) SPACE(1)
  rec
  WITH SIZE 80 BY 10 TITLE "The rectangle is 5 by 5".

ON CHOOSE OF b_size IN FRAME butt-frame
    ASSIGN rec:WIDTH-CHARS IN FRAME sz-frame =
                 RANDOM(1, FRAME sz-frame:WIDTH-CHARS - 3)
           rec:HEIGHT-CHARS = RANDOM(1, FRAME sz-frame:HEIGHT-CHARS - 2)
           FRAME sz-frame:TITLE =  "The rectangle is " +
                 STRING(rec:WIDTH-CHARS) + " by " +
                 STRING(rec:HEIGHT-CHARS).ENABLE rec WITH FRAME sz-frame.

ENABLE b_size b_quit WITH FRAME butt-frame.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame.
```

Notes

- Progress supports fractional character units. Therefore, if you express dimensions in characters, the width and height values can include up to two decimal places.
- For SpeedScript, the PIXEL options are not valid.

See also

COMBO-BOX phrase, DEFINE BROWSE statement, DEFINE BUTTON statement, DEFINE IMAGE statement, DEFINE RECTANGLE statement, EDITOR phrase, Frame phrase, RADIO-SET phrase, SELECTION-LIST phrase, SLIDER phrase

SLIDER phrase

Describes a slider representation of a field or variable. A slider is a graphical representation of a numeric range. It is composed of a rectangular area that contains a line or trackbar. A marker or pointer within the region indicates the current value. The SLIDER phrase is an option of the VIEW-AS phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
VIEW-AS SLIDER
  MAX-VALUE max-value MIN-VALUE min-value
  [ HORIZONTAL | VERTICAL ]
  [ NO-CURRENT-VALUE ]
  [ LARGE-TO-SMALL ]
  [ TIC-MARKS { NONE | TOP | BOTTOM | LEFT | RIGHT | BOTH }
    [ FREQUENCY n ] ]
  [ TOOLTIP tooltip ]
  [ size-phrase ]
```

MAX-VALUE *max-value* MIN-VALUE *min-value*

Sets the range of values for the slider. Both *max-value* and *min-value* must be integer constants. Depending on the windowing system in use, the maximum value, minimum value, or both can be displayed with the slider. If you do not specify either a minimum value or a maximum value, the default is 0. *Max-value* must be greater than *min-value*.

In Windows only, you can use the MAX-VALUE and MIN-VALUE options with the LARGE-TO-SMALL option to indicate that the slider's maximum display value displays first and the minimum value displays last as you move the slider control.

HORIZONTAL | VERTICAL

Specifies the orientation of the slider. If the orientation is VERTICAL, the slider displays with the minimum value at the bottom and the maximum value at the top. The user can then change the value by moving the trackbar up or down. If the orientation is HORIZONTAL (the default), the slider displays with the minimum value at the left and the maximum value at the right. The user can then change the value by moving the bar left or right.

NO-CURRENT-VALUE

The default is to display the current value for a given position on the slider control. The NO-CURRENT-VALUE option allows you to override this default behavior to indicate that the slider will not automatically display the current value of the slider.

For example, if the MIN-VALUE is 10, the default is to display the value 10 when the slider is first realized, and to update the displayed value whenever a user moves the slider trackbar.

The NO-CURRENT-VALUE option is supported in Windows only.

LARGE-TO-SMALL

The default numeric range that a slider can display is small (minimum) to large (maximum). The LARGE-TO-SMALL option allows you to override this default behavior as follows:

- When the slider is positioned horizontally, the left-most position on the trackbar displays the maximum value and the right-most position displays the minimum value.
- When the slider is positioned vertically, the bottom-most position on the trackbar displays the maximum value and the top-most position displays the minimum value.

The LARGE-TO-SMALL option is supported in Windows only.

`TIC-MARKS {NONE | TOP | BOTTOM | LEFT | RIGHT | BOTH}`

Enables the display of short hash marks on the outside of a slider to help indicate the movement of the trackbar with the slider widget. The default is not to display tic marks. If you specify the TIC-MARKS option, it is assumed that you are using new code to create a slider, and the trackbar on the slider widget will be relatively large.

However, if you leave the TIC-MARKS option out, the 4GL assumes that you are migrating old code, and the default size of the slider is the size originally defined for the slider in the old code.

If you want to use the large trackbar but do not want tic marks to display, specify TIC-MARKS NONE.

To implement the TIC-MARKS option, you must also specify on which side, or sides, of the trackbar tick-marks display by using the additional TOP, BOTTOM, LEFT, RIGHT, or BOTH qualifying options.

The TIC-MARKS option is supported in Windows only.

`FREQUENCY` *`n`*

Used only with the TIC-MARKS option, indicates the incremental display of the TIC-MARKS. For example, if you indicate a frequency of 5, a tic mark displays in every fifth position along the slider bar.

The FREQUENCY option is supported in Windows only.

`TOOLTIP` *`tooltip`*

Allows you to define a help text message for a text field or text variable. Progress automatically displays this text when the user pauses the mouse pointer over a text field or text variable for which a ToolTip is defined.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed. No ToolTip is the default. The TOOLTIP option is supported in Windows only.

size-phrase

Specifies the outside dimensions of the slider widget. This is the syntax for *size-phrase*:

```
{ SIZE | SIZE-CHARS | SIZE-PIXELS } width BY height
```

For more information, see the SIZE phrase reference entry.

Example

The following procedure displays a slider with tic-marks noted every tenth position, and prompts the user to pick an integer value. After the user picks an integer, the program displays in a separate frame the text "You selected" followed by the value.

r-slide.p

```
DEFINE VAR choice AS INTEGER LABEL "You selected".
DEFINE VAR a AS INTEGER.
UPDATE a VIEW-AS SLIDER
  MAX-VALUE 100 MIN-VALUE 1 SIZE-CHAR 33 BY 3
  TIC-MARKS BOTTOM FREQUENCY 10
  LABEL "Slide to select an integer. Then press GO."
  WITH FRAME f Three-D.
choice = a.
DISPLAY choice WITH FRAME b SIDE-LABELS THREE-D.
PAUSE.
```

Notes

- If the slider is too short, the user might not be able to select from the full range of values.
- If you display the slider horizontally, the *width* value determines the length of the slider, and the *height* value adds white space above and below the slider; similarly, if you display the slider vertically, the *height* value determines the length of the slider, and the *width* value adds white space on either side of the slider.
- Note that Windows allows a user to transfer focus to the slider by pressing **ALT** and one of the letters in the label.
- In character interfaces, a slider widget has a minimum width that is dependent on the specified maximum value (MAX-VALUE attribute). The minimum height for a slider widget in a character interface is 2 character units. You can specify a value as low as 1.5 character units for the height of a slider in a character interface; however, Progress rounds the value up to 2 character units.

See also

VIEW-AS phrase

SQRT function

Returns the square root (as a decimal value) of an expression you specify.

Syntax

```
SQRT ( expression )
```

expression

A numeric expression. If the value of the expression is negative, SQRT returns the Unknown value (?).

Example

This procedure prompts for a number and then displays the square root of that number:

r-sqrt.p

```
DEFINE VARIABLE num AS INTEGER FORMAT ">,>>9"
  LABEL "Enter a number between 1 and 9,999".

REPEAT WITH SIDE-LABELS CENTERED
  TITLE "SQUARE ROOT GENERATOR" COLUMN 20 1 DOWN.
  DISPLAY SKIP(2).
  SET num SKIP(2).
  DISPLAY "The square root of " + string(num) + " is"
    FORMAT "x(27)"
    SQRT(num) FORMAT ">>>.9999".
END.
```

SSL-SERVER-NAME function

Returns the digital certificate subject name for an OpenEdge database connected via SSL. If a database connection does not exist or the connection is not using SSL, this function returns the Unknown value (?).

Syntax

```
SSL-SERVER-NAME ( logical-database-name )
```

logical-database-name

A quoted character string or character expression that specifies the database by its logical name.

Example

The following example returns the digital certificate subject name of the first database to which the OpenEdge session is connected:

```
SSL-SERVER-NAME (1).
```

The following example returns the digital certificate subject name of the database with the logical name mydb:

```
SSL-SERVER-NAME (mydb).
```

See also

Server object handle, Socket object handle

STATUS statement

Specifies the text that appears in the status line of a window. Progress displays the following default messages on that line:

- When a procedure is blocked and is waiting for the user to enter data into a frame field, the status message is "Enter data or press *end-error* to end," where *end-error* is the key label for the **END-ERROR** key.
- When a procedure reaches a PAUSE statement, the status message is "Press space bar to continue."
- While procedure is not blocked for input, the status message blank.

Note: Does not apply to SpeedScript programming.

Syntax

```
STATUS
  {   DEFAULT [ expression ]
    | INPUT [ OFF | expression ]
  }
  [ IN WINDOW window ]
```

`DEFAULT` *expression*

Replaces the default status message when a user is running a procedure (the default status message is blanks). The *expression* must be character and must be enclosed in quotes if it is a constant. If you do not specify an *expression*, Progress resets the STATUS DEFAULT line to its original state. The STATUS DEFAULT is a maximum of 63 characters.

`INPUT OFF`

Tells Progress not to display an input status message.

`INPUT` *expression*

Replaces the default status message when a user is entering data into a frame field. The *expression* must be character and must be enclosed in quotes if it is a constant. If you do not specify an *expression*, Progress resets the STATUS INPUT line to its original state.

IN WINDOW *window*

Specifies the window in which to set the status message. If you omit the IN WINDOW phrase, the STATUS statement applies to the current window.

Example

This procedure replaces the default status messages with two other messages:

r-status.p

```
STATUS DEFAULT "All Around Sports Order Processing System".
STATUS INPUT "Enter data, or use the " + KBLABEL("END-ERROR") +
  " key to exit".
FOR EACH customer:
  DISPLAY name.
  FOR EACH order OF customer:
    UPDATE order-num promise-date order-date ship-date.
  END.
  UPDATE credit-limit.
END.
```

Notes

- After you use the STATUS DEFAULT, STATUS INPUT OFF, or STATUS INPUT statement during a session, that statement is in effect for all the procedures run in that session, unless it is overridden by other STATUS statements in those procedures, or until you return to the Procedure Editor.
- You cannot use the STATUS statement to change the default status messages displayed while you are in the Procedure Editor.
- You can use the PAUSE statement to override the default status message displayed when Progress encounters a PAUSE statement.
- When you use the HELP attribute to display help text for a widget, Progress overwrites the status text with the HELP text.

See also

MESSAGE statement, PAUSE statement

STOP statement

Signals the STOP condition in the current block. By default, the STOP condition stops processing a procedure, backs out the active transaction, and returns to the startup procedure or to the Progress Editor. You can change this behavior by including the ON STOP phrase on a block header.

Syntax

```
STOP
```

Examples

In any procedure, the outermost block that updates the database is the system transaction. In this procedure, the first iteration of the FOR EACH block starts a system transaction. The transaction ends when that iteration ends. Another transaction starts at the start of the next iteration. After you update the credit-limit field, Progress prompts you to STOP. If you enter yes, the STOP statement stops the procedure and undoes any database modifications made in that transaction.

r-stop.p

```
DEFINE VARIABLE ans AS LOGICAL.

FOR EACH customer:
  DISPLAY cust-num name.
  UPDATE credit-limit.
  ans = no.
  MESSAGE "Stopping now undoes changes to this record.".
  MESSAGE "Do you want to stop now?" UPDATE ans.
  IF ans THEN STOP.
END.
```

When you add the ON STOP phrase to the block statement of the previous procedure, it changes the default behavior of the STOP statement. In this procedure, Progress allows you to re-enter the record when you choose to stop.

r-stop2.p

```
DEFINE VARIABLE ans  AS LOGICAL.
FOR EACH customer ON STOP UNDO, RETRY:
  DISPLAY cust-num name.
  UPDATE credit-limit.
  ans = NO.
  MESSAGE "Stopping now undoes changes to this record."
    "Do you want to stop now?" VIEW-AS ALERT-BOX QUESTION
    BUTTONS YES-NO UPDATE ans.
  IF ans THEN STOP.
END.
```

Notes

- Unless you have coded an ON STOP phrase, the STOP statement stops all currently active procedures.
- If you use the Startup Procedure (-p) parameter to start the OpenEdge session, and if the startup procedure is still active, the default STOP action restarts the procedure.
- A terminal user can initiate the STOP condition by pressing **STOP**. This is usually mapped to **CTRL+BREAK** (Windows) or **CTRL+C** (UNIX). The actual mapping depends on your terminal and system configuration.

See also ON STOP phrase

STRING function

Converts a value of any data type into a character value.

Syntax

```
STRING ( source [ , format ] )
```

source

An expression of any data type that you want to convert to a character value.

format

The format you want to use for the new character value. This format must be appropriate to the data type of *source*. If you do not use this argument, Progress uses the EXPORT format for all data types (except DATETIME and DATETIME-TZ, in which case it uses the default display format). This is useful if you want to produce left-justified numbers. See *OpenEdge Development: Progress 4GL Handbook* for information on data formats.

Example

In the example procedure, the TIME function returns the number of seconds since midnight. The first DISPLAY statement in this procedure uses the STRING function to convert that value into hours and minutes. TIME is the value and “HH:MM AM” is the format used to display the result of the STRING function.

The second DISPLAY statement displays some customer information. It uses the concatenation (+) operator to join together the values of the city, state, and postal-code fields. If these fields were not joined together, the spacing would be different for each customer address depending on the length of the city name.

r-string.p

```
DISPLAY SKIP(2) "The time is now" STRING(TIME,"HH:MM AM") SKIP(2)
        WITH NO-BOX NO-LABELS CENTERED.

FOR EACH customer:
  DISPLAY name + "  --" + STRING(cust-num, ">>>9") FORMAT "x(30)" AT 1
    address AT 33
    city + ", " + state + " " + postal-code FORMAT "x(22)" AT 33
    SKIP(1)
    WITH NO-BOX NO-LABELS CENTERED.
END.
```

When you concatenate character fields, Progress creates a new character field, at least for the duration of the procedure. The default display format for character expressions such as that resulting from the concatenation is x(8). This means that Progress allows only 8 spaces for displaying the concatenation of the city, state, and postal-code fields. The FORMAT x(22) option overrides that default x(8) format, telling Progress to set aside 22 spaces for displaying the concatenation of the city, state, and postal-code fields.

Notes

- The STRING function is double-byte enabled. The *source* argument can contain double-byte data.
- If *source* is an integer and *format* begins HH:MM or HH:MM:SS, STRING formats the *source* as a time. If the hour is greater than or equal to 12 and there is an A or an a in *format*, STRING subtracts 12 from the hour and converts the A or the a to a P or p (for A.M. and P.M.). The hour 0 is treated as 12 a.m., and noon is treated as 12 p.m. If you use AM/PM format, HH is replaced by a leading blank and a digit if the hour is between 0 and 9.

  If seconds (SS) are not in the format, then the time is truncated to hours and minutes.

- If *source* is a RAW value, you must specify an appropriate *format* to return the character string representation.
- When *source* is a DATETIME or DATETIME-TZ expression, the STRING function converts the expression to a character value in the specified format. If *source* is a DATETIME expression, and a time zone offset is present in the format string, the character value contains the time zone offset of the session. If *source* is a DATETIME-TZ expression, and time zone offset is not present in the format string, the character value contains the local date and time relative to the time zone of the DATETIME-TZ value.
- The STRING function converts a DATE, and the date part of a DATETIME or DATETIME-TZ, using the format specified by the DATE-FORMAT attribute or the Date Format (-d) startup parameter.

  For more information about the Date Format (-d) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.
- You can use the STRING function to convert an object reference for a class object instance to a character value. The STRING function implicitly calls the ToString( ) method of the class to convert the specified object reference.

See also DECIMAL function, INTEGER function

SUBSCRIBE statement

Creates a subscription to a Progress named event.

Note: Progress named events are completely different from the key function, mouse, widget, and direct manipulation events described in the "Events Reference" section on page 2171. For more information on Progress named events, see *OpenEdge Development: Progress 4GL Handbook*.

Syntax

```
SUBSCRIBE [ PROCEDURE subscriber-handle ]
  [ TO ] event-name
  { IN publisher-handle | ANYWHERE }
  [ RUN-PROCEDURE local-internal-procedure ]
  [ NO-ERROR ]
```

PROCEDURE *subscriber-handle*

A procedure or handle representing the subscriber.

The PROCEDURE option lets one procedure create a subscription on behalf of another. For example, if you want procedure A to create a subscription on behalf of procedure B, set *subscriber-handle* to the procedure handle of B.

If the PROCEDURE option does not appear, Progress creates a subscription on behalf of THIS-PROCEDURE, the procedure that contains the SUBSCRIBE statement.

TO *event-name*

A quoted string or a character expression representing the name of the event.

IN *publisher-handle*

Subscribes to the named events published by *publisher-handle*.

If *publisher-handle* is not a valid procedure or widget handle at the time the SUBSCRIBE statement executes, Progress reports a run time error unless you specify the NO-ERROR option.

ANYWHERE

Subscribes to named events published within the OpenEdge session-regardless of the publisher.

RUN-PROCEDURE *local-internal-procedure*

A quoted string or character expression representing the name of an internal procedure that resides within the subscribing program. Progress runs *local-internal-procedure* when the named event occurs.

If the RUN-PROCEDURE option does not appear, when the named event occurs, Progress runs an internal procedure with the same name as the named event.

Note: The RUN-PROCEDURE option lets you create a subscription when the event name and the procedure name do not match, or when you must subscribe to two different events that have the same name.

When the named event occurs, Progress RUNs each subscriber's local internal procedure, passing the parameters, if any, The order in which Progress notifies subscribers is undefined. Progress always performs this RUN with an implicit NO-ERROR, and logs errors to the ERROR-STATUS system handle.

NO-ERROR

Tells Progress not to report a run time error if *publisher-handle* or *subscriber-handle* is not a valid procedure handle, or if Progress cannot evaluate an *event-name* expression. Errors are still generated, however, and are stored in the ERROR-STATUS handle.

Example

For an example, see the reference entry for the PUBLISH statement in this manual.

Notes

- Within the local internal procedure, you can get a handle to the publisher of the named event by using the SOURCE-PROCEDURE system handle. For more information on the SOURCE-PROCEDURE system handle, see its reference entry in this book.
- If Progress detects a redundant SUBSCRIBE statement—that is, a SUBSCRIBE statement with the same event name, and either the same publisher handle or the same ANYWHERE option—Progress does not report an error.
- If *event-name* is a string containing spaces or is otherwise not a standard Progress name, use one of the following techniques:
  - Use the RUN-PROCEDURE option to assign the local internal procedure a more conventional name.
  - When you define *local-internal-procedure*, put its name in quotes, as in the following example:

```
PROCEDURE "spaced event":
```

See also PUBLISH statement, PUBLISHED-EVENTS attribute, UNSUBSCRIBE statement

SUBSTITUTE function

This function returns a character string that is made up of a base string plus the substitution of arguments in the string. It allows you to use a single string in place of concatenated strings. It is designed to simplify the task of translating an application from one language to another. This function is similar to the `sprintf` function of the C programming language.

Syntax

```
SUBSTITUTE ( base-string [ , arg ] ... )
```

base-string

A character string optionally containing substitution parameters of the form &*n*, where *n* is an integer between 1 and 9, inclusive.

arg

A constant, field name, variable, or expression that results in a character string value. These argument values replace substitution parameters in *base-string*.

Examples

These statements display the same message:

```
MESSAGE SUBSTITUTE("There were &1 records in &2 tables",
  rec-count, table-count).
```

```
MESSAGE "There were" rec-count "records in"
  table-count "tables".
```

You can alter the position of the substitution parameters, as in this statement.

```
SUBSTITUTE("&2 comes before &1", "Friday", "Monday").
```

Notes

- The SUBSTITUTE function is double-byte enabled. The specified *base-string* and *arg* values can contain double-byte characters.
- To include an ampersand character in *base-string*, enter two ampersands (&&).
- The character following the ampersand character must be a digit, or Progress returns a run-time error.
- To display the result of the SUBSTITUTE function in a frame, you must specify FORMAT or accept the default format of X(8).
- If you use a substitution parameter in *base string* but do not specify a corresponding argument, Progress replaces the substitution parameter with an empty string.
- Any substitution parameter can appear multiple times in *base string*. For example:

```
phrase = "finish on time".
DISPLAY SUBSTITUTE("When I say &1, I mean &1!", phrase) FORMAT "X(70)".
```

The code above displays the following line:

```
When I say finish on time, I mean finish on time!
```

See also REPLACE function

SUBSTRING function

Extracts a portion of a character string from a field or variable.

Syntax

```
SUBSTRING (source , position [ , length [ , type ] ] )
```

source

A CHARACTER or LONGCHAR expression from which you want to extract characters or bytes.

position

An INTEGER expression that indicates the position of the first character you want to extract from *source*.

length

An INTEGER expression that indicates the number of characters you want to extract from *source*. If you do not use the *length* argument or specify -1 as the length, SUBSTRING uses the remainder of the string from the specified *position*.

type

A CHARACTER expression that directs Progress to interpret the specified *position* and *length* values as character units, bytes, or columns. A double-byte character registers as one character unit. By default, Progress interprets the specified *position* and *length* values as character units.

There are FOUR valid types: "CHARACTER," "FIXED," "COLUMN," and "RAW." The expression "CHARACTER" specifies character units. The expression "FIXED" specifies that *position* is in character units and the length is in bytes, but directs SUBSTRING to yield only whole characters. That is, if the last byte or bytes represent part of, but not all of, a multi-byte character, these bytes are excluded. The expression "COLUMN" specifies display or print character-columns. The expression "RAW" specifies bytes. If you specify the *type* as a constant expression, Progress validates the type specification at compile time. If you specify the *type* as a non-constant expression, Progress validates the type specification at run time.

Note: If *source* is a LONGCHAR expression, "CHARACTER" is the only valid type and the default type.

Example

The `r-substr.p` procedure uses the SUBSTRING function to create invoice numbers. You supply a starting invoice number. The first SUBSTRING function produces the first two characters of today's date; the second SUBSTRING function produces the last two characters of today's date. The procedure concatenates these four characters to a hyphen and the number you entered to produce an invoice number.

r-substr.p

```
DEFINE VARIABLE inv-num AS CHARACTER FORMAT "x(11)"
  LABEL "Invoice Number".
DEFINE VARIABLE snum AS INTEGER FORMAT "9999"
  LABEL "  Starting Order Number".
DEFINE VARIABLE enum LIKE snum LABEL "    Ending Order Number".
DEFINE VARIABLE num LIKE snum LABEL "Starting Invoice Number".

UPDATE "      Creating Invoices"
  SKIP(2) snum SKIP(1) enum SKIP(2) num SKIP(2)
  WITH SIDE-LABELS CENTERED NO-BOX.

FOR EACH order WHERE order.order-num >= snum
  AND order.order-num <= enum:
  inv-num =  SUBSTRING(STRING(TODAY),1,2,"CHARACTER") +
             SUBSTRING(STRING(TODAY),7,2,"CHARACTER") + " - " +
              STRING(num,"9999").
  DISPLAY order-num inv-num WITH CENTERED.
  /* Do creation and printing of invoice here */

  num = num + 1.
END.
```

See also

OVERLAY statement, SUBSTRING statement

SUBSTRING statement

Replaces characters in a field or variable with an expression you specify.

Syntax

```
SUBSTRING
  ( source , position [ , length [ , type ] ] ) = expression
```

source

A field or variable of type CHARACTER or LONGCHAR in which you want to store the specified *expression*.

position

An integer expression that indicates the position in *source* in which you want to start storing *expression*. If *position* is longer than *source*, Progress pads *source* with blanks to equal the length of *position*.

length

An integer expression that indicates the number of positions you want to replace in *source*. If you specify a length of 0, the expression is inserted at the *position* and everything else moves to the right. If you do not use the *length* argument or specify -1 as the length, SUBSTRING puts the entire *expression* into *source*, replacing everything in *source* necessary to make room for *expression*.

type

A character expression that directs Progress to interpret the specified *position* and *length* values as character units, bytes, or columns. A double-byte character registers as one character unit. By default, Progress interprets the specified *position* and *length* values as character units.

There are three valid types: "CHARACTER," "RAW,".and "COLUMN." The expression "CHARACTER" specifies character units. The expression "RAW" specifies bytes. The expression "COLUMN" specifies display or print character-columns. If you specify the type as a constant expression, Progress validates the type specification at compile time. If you specify the type as a non-constant expression, Progress validates the type specification at run time.

Note: If *source* is a LONGCHAR expression, "CHARACTER" is the only valid type and the default type.

expression

A constant, field name, variable name, or expression of type CHARACTER or LONGCHAR that results in a character string whose value you want to store in *source*. Progress does not pad or truncate *expression*.

Example

The `r-sub.p` procedure uses the SUBSTRING statement to replace a segment of text with the expression in the SUBSTRING statement XXXXXXXXX. The procedure first displays the text you can work with in the Original Text frame. Then the procedure prompts you for the start position of the replacement and the length of the replacement. Under the WORD heading, you see the revised text.

r-sub.p

```
DEFINE VARIABLE rtext AS CHARACTER FORMAT "x(50)" .
DEFINE VARIABLE orig AS CHARACTER FORMAT "x(31)".
DEFINE VARIABLE strt AS INTEGER FORMAT ">9".
DEFINE VARIABLE leng AS INTEGER FORMAT ">9".

orig = "Now is the time to use PROGRESS".
DISPLAY orig WITH CENTERED TITLE "Original Text" NO-LABEL.
REPEAT:
  rtext = orig.
  UPDATE strt LABEL "START" leng LABEL "LENGTH".
  SUBSTRING(rtext,strt,leng,"CHARACTER") = "XXXXXXXXX".
  DISPLAY rtext LABEL "WORD" WITH CENTERED.
END.
```

Notes

- When you use the SUBSTRING statement, the length of the target string may change (because the length of the expression value is not the same as the length of the substring you are replacing). By contrast, the OVERLAY statement never changes the length of the target string. The OVERLAY statement truncates or pads the expression value to make it the same length as the substring to be replaced.
- Do not split double-byte characters. This statement allows you to replace either the lead- or trail-byte of the target string when you specify "RAW" for the *type* parameter.

See also OVERLAY statement, SUBSTRING function

SUPER function

Runs the super version of the current user-defined function.

This language element must appear within a user-defined function, but can appear anywhere within the user-defined function. If this language element does not appear within a user-defined function, the compiler reports an error.

Syntax

```
SUPER [ ( parameter [ , parameter ] ... ) ]
```

parameter

A parameter of the super version of the current user-defined function. These parameters must have the same signature (number of parameters, and type and mode of each) as the parameters of the current user-defined function. You can, however, adjust a parameter's value.

For the *parameter* syntax, see the Parameter definition syntax reference entry in this book.

If a user-defined function can not be located in any super procedure, Progress generates the following error message:

```
SUPER version of user-defined function name invoked but could not be found
```

Errors are stored in the ERROR-STATUS handle when NO-ERROR is specified.

Example

For an example of the SUPER function, see the RUN SUPER statement reference entry in this book.

Notes

- To run the super version of an internal procedure, use the RUN SUPER statement.
- For the rules that Progress uses to find the super version of the current user-defined function, see the ADD-SUPER-PROCEDURE( ) method reference entry in this book.
- For an overview of super procedures, see *OpenEdge Development: Progress 4GL Handbook*.

See also

ADD-SUPER-PROCEDURE( ) method, REMOVE-SUPER-PROCEDURE( ) method, RUN SUPER statement, SOURCE-PROCEDURE system handle, SUPER-PROCEDURES attribute, TARGET-PROCEDURE system handle

SUPER system reference

A system reference that lets a subclass access the PUBLIC and PROTECTED methods of its super class in the inherited class hierarchy. If the specified method definition is not found in the subclass's immediate super class, Progress repeatedly looks to the next super class in the inherited class hierarchy until it finds the definition.

Syntax

```
SUPER:method-name ( [ parameter [, parameter ] ... ] )
```

method-name

The name of a method defined in a super class.

(*parameter* [, *parameter*] ...)

Specifies one or more parameters to pass to the method. You must provide the parameters identified by the method, and the parameters must match with respect to the number, data type, and mode.

For the parameter passing syntax, see the Parameter passing syntax reference entry in this book.

Note

You typically use the SUPER system reference within a method of a class in the hierarchy to invoke a method defined in a super class that was overridden in a subclass. For more information about using the SUPER system reference, see *OpenEdge Getting Started: Object-oriented Programming*.

SYSTEM-DIALOG COLOR statement (Windows only; Graphical interfaces only)

Displays a dialog box that lets the user choose and associate a system color with the specified dynamic color number. The SYSTEM-DIALOG COLOR statement provides a dialog box appropriate to the graphical environment in which it runs.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG COLOR color-number
  [ UPDATE logical-variable ]
  [ IN WINDOW window ]
```

color-number

An integer expression that evaluates to a Progress color number from 0 to 255, inclusive, that is defined as dynamic through the SET-DYNAMIC method of the COLOR-TABLE handle. The color dialog associates the Progress color specified by *color-number* with the system color value the user selects in the dialog box. The user chooses the OK button to confirm the choice. The user can close the dialog box without changing the color by choosing the Cancel button.

UPDATE *logical-variable*

Specifies a logical variable to return the status of the user's color dialog interaction. If the user chooses the OK button, the dialog sets *logical-variable* to TRUE. If the user chooses the Cancel button, the dialog sets *logical-variable* to FALSE.

IN WINDOW *window*

Specifies the window where the dialog box is displayed. The value *window* must be the handle of a window.

Example

The following procedure displays a dialog box that allows the user to assign new foreground and background colors to the dialog box. A radio set in the dialog box lists selections for foreground and background that correspond to the numbers nine and eight, respectively. Choosing the OK button opens a color dialog box to assign a new system color to the selected color number. Note that the UPDATE option is not used to return a termination status because the dialog does not require the user to select a new color; it only provides the option. The procedure terminates when the user chooses the Cancel button in the radio selection dialog box.

r-coldlg.p

```
DEFINE VARIABLE front-color  AS INTEGER INITIAL 9.
DEFINE VARIABLE back-color   AS INTEGER INITIAL 8.
DEFINE VARIABLE curr-color   AS INTEGER INITIAL 9
              VIEW-AS RADIO-SET
                  RADIO-BUTTONS "Foreground", 9,
                                "Background", 8.
DEFINE BUTTON ok-button     LABEL "OK".
DEFINE BUTTON cancel-button LABEL "Cancel" AUTO-ENDKEY.

FORM
    SKIP(0.5) SPACE(0.5)
    curr-color SPACE(2) ok-button SPACE(2) cancel-button
    SPACE(0.5) SKIP(0.5)
    WITH FRAME color-frame NO-LABELS
    TITLE "Choose frame colors ..."
    FGCOLOR front-color
    BGCOLOR back-color
    VIEW-AS DIALOG-BOX.

ON CHOOSE OF ok-button IN FRAME color-frame
DO:
   ASSIGN curr-color.
   IF NOT COLOR-TABLE:GET-DYNAMIC(curr-color) AND
      NOT COLOR-TABLE:SET-DYNAMIC(curr-color,TRUE)
   THEN MESSAGE "Color must be DYNAMIC to edit.".
   ELSE SYSTEM-DIALOG COLOR curr-color.
END.

UPDATE curr-color ok-button cancel-button WITH FRAME color-frame.
```

Note that the trigger for the ok-button must assign the curr-color variable to obtain the latest value selected for the radio set. The GET-DYNAMIC and SET-DYNAMIC methods are used to ensure that the color is dynamic before modifying it.

Notes

- For more information on defining dynamic colors, see *OpenEdge Deployment: Managing 4GL Applications*.
- Use the *color-number* in a COLOR phrase to assign the selected color to a widget. For more information on assigning colors to widgets, see *OpenEdge Development: Progress 4GL Handbook*.

See also COLOR phrase, COLOR-TABLE system handle

SYSTEM-DIALOG FONT statement (Windows only; Graphical interfaces only)

Displays a dialog box that allows the user to select and associate a system font with the specified font number. The SYSTEM-DIALOG FONT statement provides a dialog box appropriate to the graphical environment in which it runs.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG FONT font-number
  [ ANSI-ONLY ]
  [ FIXED-ONLY ]
  [ MAX-SIZE point-size ]
  [ MIN-SIZE point-size ]
  [ UPDATE logical-variable ]
  [ IN WINDOW window ]
```

font-number

An integer expression that returns a Progress font number (0 to 255), inclusive, which is defined in the setup file for your environment. The font dialog associates the Progress font specified by *font-number* with the system font the user selects in the dialog. The user confirms the selection and completes the dialog by choosing the OK button. The user interrupts the dialog without changing the font by choosing the Cancel button.

ANSI-ONLY

Allows the font dialog to provide only fonts that contain character representations and that do not include graphic symbols.

FIXED-ONLY

Allows the font dialog to provide only mono-spaced fonts.

MAX-SIZE *point-size*

Has no effect; supported only for backward compatibility.

MIN-SIZE *point-size*

Has no effect; supported only for backward compatibility.

UPDATE *logical-variable*

Specifies a logical variable to return the status of the user's font dialog interaction. If the user clicks on the OK button, the dialog sets *logical-variable* to TRUE. If the user chooses on the Cancel button, the dialog sets *logical-variable* to FALSE.

IN WINDOW *window*

Specifies the window from which the dialog box is displayed. The value *window* must be the handle of a window.

Example

The following procedure displays a dialog box that allows the user to change the font of either its radio set or its buttons. The radio set lists a font number for each selection: font 1 for the radio set and font 2 for the buttons. Choosing on the OK button opens a font dialog to assign a new system font to the font number selected in the radio set. Note that the UPDATE option is not used to return a termination status because the dialog does not require the user to select a new font; it only provides the option. The procedure terminates when the user chooses the Cancel button.

r-fntdlg.p

```
DEFINE VARIABLE Font1      AS INTEGER INITIAL 1.
DEFINE VARIABLE Font2      AS INTEGER INITIAL 2.
DEFINE VARIABLE FontSelect AS INTEGER INITIAL 1
    VIEW-AS RADIO-SET
        RADIO-BUTTONS "Font 1", 1, "Font 2", 2
        FONT Font1.
DEFINE BUTTON bOK     LABEL "OK" FONT Font2.
DEFINE BUTTON bCANCEL LABEL "Cancel" FONT Font2 AUTO-ENDKEY.

FORM
    SKIP(0.5) SPACE(0.5)
    FontSelect SPACE(2) bOK SPACE(2) bCANCEL
    SPACE(0.5) SKIP(0.5)
    WITH FRAME fFont TITLE "Choose frame fonts ..." VIEW-AS DIALOG-BOX.

ON CHOOSE OF bOK IN FRAME fFont
DO:
    IF INTEGER(FontSelect:SCREEN-VALUE IN FRAME fFont) = Font1 THEN
        SYSTEM-DIALOG FONT Font1.
    ELSE
        SYSTEM-DIALOG FONT Font2.
END.

UPDATE FontSelect bOK bCANCEL WITH FRAME fFont.
```

Note: The CHOOSE OF OK event trigger must reference the SCREEN-VALUE attribute of the FontSelect variable to obtain the latest value selected for its radio set. This is because the UPDATE statement has not yet completed during the event, and has not updated the FontSelect record buffer from the frame buffer. The initial value of FontSelect is its value in the record buffer immediately before the UPDATE statement executes.

Notes

- For more information on defining font numbers, see *OpenEdge Deployment: Managing 4GL Applications*.
- Use the *font-number* with the FONT option to assign the selected font to a widget. For more information on assigning fonts to widgets, see *OpenEdge Development: Progress 4GL Handbook*.

SYSTEM-DIALOG GET-DIR statement (Windows only)

Displays a dialog box that allows the user to enter a directory name that is assigned to a character variable. The SYSTEM-DIALOG GET-DIR statement provides a dialog box appropriate to the environment in which it runs.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG GET-DIR character-field
  [ INITIAL-DIR directory-string ]
  [ RETURN-TO-START-DIR ]
  [ TITLE title-string ]
```

character-field

The character field or variable that contains the directory name the user enters. The user can enter the directory name by typing it or selecting it from a list of directories in the common dialog directory. The user confirms the entry and completes the dialog by choosing the OK button. The user can interrupt the dialog without any selection by choosing the Cancel button.

INITIAL-DIR *directory-string*

Sets the starting directory for this invocation of SYSTEM-DIALOG GET-DIR to the pathname specified in *directory-string* before starting the dialog. The *directory-string* is a character expression that must evaluate to a valid pathname in your environment. The default starting directory is either the current working directory or the directory left from the last invocation of SYSTEM-DIALOG GET-DIR.

RETURN-TO-START-DIR

This option resets the current directory to the starting directory when the common dialog ends. This is the directory specified by the INITIAL-DIR option or the default starting directory.

If you do not specify this option, the directory remains set at the last directory referenced by the user. This directory becomes the default initial directory for subsequent invocations of SYSTEM-DIALOG GET-DIR. This option also has no effect on subsequent invocations that specify the INITIAL-DIR option.

TITLE *title-string*

Specifies a title for the dialog box. The value *title-string* can be any character expression. If you do not specify a title, the dialog uses the system default for your environment.

Notes

- The default common dialog directory for the initial invocation of SYSTEM-DIALOG GET-DIR is the current working directory. You can specify a different starting common dialog directory with the INITIAL-DIR option and the user can change the common dialog directory by referencing a different directory in the common dialog.
- The Windows common dialog never searches the PROPATH, and always returns the full pathname of the entered relative pathname appended to the current common dialog directory.

SYSTEM-DIALOG GET-FILE statement (Windows only)

Displays a dialog box that allows the user to enter a filename that is assigned to a character variable. The SYSTEM-DIALOG GET-FILE statement provides a dialog box appropriate to the environment in which it runs.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG GET-FILE character-field
  [   FILTERS name filespec
      [ , name filespec ] ...
      [ INITIAL-FILTER filter-num ]
  ]
  [ ASK-OVERWRITE ]
  [ CREATE-TEST-FILE ]
  [ DEFAULT-EXTENSION extension-string ]
  [ INITIAL-DIR directory-string ]
  [ MUST-EXIST ]
  [ RETURN-TO-START-DIR ]
  [ SAVE-AS ]
  [ TITLE title-string ]
  [ USE-FILENAME ]
  [ UPDATE logical-variable ]
  [ IN WINDOW window ]
```

character-field

The character field or variable that contains the filename the user enters. The user can enter the filename by typing it or selecting it from a list of files in the common dialog directory. The user confirms the entry and completes the dialog by choosing the OK button. The user can interrupt the dialog without any selection by choosing the Cancel button.

You can also use *character-field* to pass a default filename entry to the dialog. See the USE-FILENAME option for more information.

FILTERS *name filespec*

Defines one or more filters for the filename dialog. Each filter selects a subset of the available files in the common dialog directory to build the dialog file selection-list. A filter consists of two parts: a label (*name*) and file specification (*filespec*).

The *name* is a character expression used as a label for your filter. Windows uses the label to identify the filter in a filter selection-list. The user can select the label to view the list of files selected by the filter.

The *filespec* is a character expression that evaluates to a file specification string. This string can consist of any wild cards or regular expressions used to generate valid file specifications in your environment. In Windows, *filespec* can also consist of multiple file specifications, separating each one with a comma, for example: *.p,*.i,*.r.

If you do not specify any filters, the dialog builds the selection-list with all files in the directory.

INITIAL-FILTER *filter-num*

Specifies the initial filter list defined by the FILTERS option, where *filter-num* is an integer expression that evaluates to the position of the filter in the list, starting from 1.

If you do not specify the INITIAL-FILTER option, the dialog uses the first filter in the list as the initial filter.

ASK-OVERWRITE

Causes a the dialog to prompt for confirmation if the user enters the name of a file that already exists. By default, the dialog does not prompt for confirmation if the user enters an existing filename. In Windows, this option is ignored unless SAVE-AS is also specified.

CREATE-TEST-FILE

Causes the filename dialog to create a temporary file before it completes in order to verify that the user has write access to the directory path specified for the filename entry. If the dialog cannot write the file, it displays an error message and prompts for another filename entry. The dialog does not complete until the user enters a filename associated with a writable directory or chooses the Cancel button to interrupt the dialog. After successful completion, the dialog deletes the temporary file.

This option is especially appropriate with the SAVE-AS option to verify the ability to save a file.

DEFAULT-EXTENSION *extension-string*

Specifies a default extension (or suffix) to be appended to the user's filename entry after completing the filename dialog, where *extension-string* is a character expression that evaluates to a valid file extension in your environment, including all required punctuation. In Windows, the extension must start with a period.

The Windows dialog appends the specified extension to the user's filename entry only if the entry does not already contain an extension.

INITIAL-DIR *directory-string*

Sets the starting directory for this invocation of SYSTEM-DIALOG GET-FILE to the pathname specified in *directory-string* before starting the dialog. The *directory-string* is a character expression that must evaluate to a valid pathname in your environment. The default starting directory is either the current working directory or the directory left from the last invocation of SYSTEM-DIALOG GET-FILE.

MUST-EXIST

Requires that the user's filename entry, complete with any specified default extension, must exist in the directory specified for the filename entry before the dialog completes. If it does not exist, the dialog displays an error message and prompts for another filename entry. The dialog does not complete until the user enters the name of an existing file or chooses the Cancel button to interrupt the dialog.

RETURN-TO-START-DIR

This option resets the current directory to the starting directory when the common dialog ends. This is the directory specified by the INITIAL-DIR option or the default starting directory.

If you do not specify this option, the directory remains set at the last directory referenced by the user. This directory becomes the default initial directory for subsequent invocations of SYSTEM-DIALOG GET-FILE. This option also has no effect on subsequent invocations that specify the INITIAL-DIR option.

SAVE-AS

Causes the dialog box to become a Save As dialog box. For a Save As dialog box, the default box title is "Save As". You can use the ASK-OVERWRITE option with SAVE-AS to get confirmation before accepting an existing file from the dialog.

TITLE *title-string*

Specifies a title for the dialog box. The value *title-string* can be any character expression. If you do not specify a title, the dialog uses the system default for your environment.

USE-FILENAME

Specifies the contents of *character-field* as the default filename entry for the dialog.

During the dialog, the user can accept the default entry or override it by entering or selecting another filename.

UPDATE *logical-variable*

Specifies a logical variable to return the status of the user's filename dialog interaction. If the user chooses the OK button, the dialog sets *logical-variable* to TRUE. If the user chooses the Cancel button, the dialog sets *logical-variable* to FALSE.

IN WINDOW *window*

Specifies the window from which the dialog box is displayed. The value *window* must be the handle of a window.

Example

The following example uses the filename dialog box to run procedures. It allows the user to select and run procedure files until they choose the Cancel button:

r-fildlg.p

```
DEFINE VARIABLE procname AS CHARACTER NO-UNDO.
DEFINE VARIABLE OKpressed AS LOGICAL INITIAL TRUE.

Main:
REPEAT:
    SYSTEM-DIALOG GET-FILE procname
        TITLE      "Choose Procedure to Run ..."
        FILTERS    "Source Files (*.p)"   "*.p",
                   "R-code Files (*.r)"   "*.r"
        MUST-EXIST
        USE-FILENAME
        UPDATE OKpressed.

    IF OKpressed = TRUE THEN
        RUN VALUE(procname).
    ELSE
        LEAVE Main.
END.
```

Notes

- The default common dialog directory for the initial invocation of SYSTEM-DIALOG GET-FILE is the current working directory. You can specify a different starting common dialog directory with the INITIAL-DIR option and the user can change the common dialog directory by referencing a different directory in the common dialog.
- The Windows common dialog never searches the PROPATH, and always returns the full pathname of the entered relative pathname appended to the current common dialog directory.

SYSTEM-DIALOG PRINTER-SETUP statement (Windows only)

Displays the Windows Print dialog box and lets the user set the default print context for subsequent print jobs in Windows.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-DIALOG PRINTER-SETUP
  [ NUM-COPIES expression ]
  [ LANDSCAPE | PORTRAIT ]
  [ UPDATE status ]
  [ IN WINDOW window ]
```

NUM-COPIES *expression*

Specifies the initial value of the Copies field in the Print dialog box. The value *expression* must evaluate to an integer expression. The user can change this value within the dialog box. This option is supported only with printer drivers that support multi-copy printing. Otherwise, the Copies field is disabled.

LANDSCAPE

Specifies the initial value of the Orientation field in the Properties dialog box as landscape. The user can change this value within the dialog box. The Properties dialog box is accessible from the Print dialog box. This option is supported only with printer drivers that support landscape page orientation.

PORTRAIT

Specifies the initial value of the Orientation field in the Properties dialog box as portrait. The user can change this value within the dialog box. The Properties dialog box is accessible from the Print dialog box. This option is supported only with printer drivers that support portrait page orientation.

UPDATE *status*

Specifies a logical variable to return the status of the user's dialog interaction. If the user chooses the OK button, the dialog sets *status* to TRUE. If the user chooses the Cancel button, the dialog sets *status* to FALSE.

IN WINDOW *window*

Specifies the window from which the Print dialog box is displayed. The value *window* must be the handle of a window.

Example

This example presents a dialog box that allows you to set up and print information from the sports database. When you choose the Printer Setup button, it displays the Windows Print dialog box. Using the latest settings, you can then print a list of customer names from the sports database in alphabetical order by choosing the Print Customer Names button.

r-prtdlg.p

```
DEFINE BUTTON bprintset LABEL "Printer Setup".
DEFINE BUTTON bprintnames LABEL "Print Customer Names".
DEFINE BUTTON bcancel LABEL "Cancel".
DEFINE FRAME PrintFrame
    bprintset bprintnames bcancel
WITH TITLE "Quick Printer" VIEW-AS DIALOG-BOX.

ON CHOOSE OF bprintset DO:
    SYSTEM-DIALOG PRINTER-SETUP.
END.

ON CHOOSE OF bprintnames DO:
    OUTPUT TO PRINTER.
    FOR EACH customer BY name:
        DISPLAY name WITH STREAM-IO.
    END.
    OUTPUT CLOSE.
END.

ENABLE ALL WITH FRAME PrintFrame.
WAIT-FOR CHOOSE OF bcancel IN FRAME PrintFrame.
```

Notes

- The default print context is the set of values that defines the default printer and setup for that printer in Windows. If there is no default print context, Progress uses the printer control settings from the current environment.
- Use the PRINTER-NAME attribute of the SESSION system handle to set the printer name in the default print context without user intervention.
- By default, the OUTPUT TO PRINTER statement prints jobs based on the default print context. However, you can use the OUTPUT TO PRINTER statement with its various options to override the default print context for a specific print job.

See also

OUTPUT TO statement, SESSION system handle

SYSTEM-HELP statement (Windows only)

The SYSTEM-HELP statement calls the Microsoft Windows Help engine to display Windows Help topics, and the HTML Help engine to display HTML Help topics.

Note: Does not apply to SpeedScript programming.

Syntax

```
SYSTEM-HELP file-string
  [ WINDOW-NAME window-name ]
  {   CONTENTS
    | CONTEXT int-expr
    | HELP-TOPIC string
    | KEY string
    | ALTERNATE-KEY string
    | POSITION X x Y y WIDTH dx HEIGHT dy
    | POSITION MAXIMIZE
    | QUIT
    | SET-CONTENTS int-expr
    | CONTEXT-POPUP int-expr
    | PARTIAL-KEY string
    | MULTIPLE-KEY char TEXT string
    | COMMAND string
    | FINDER
    | FORCE-FILE
    | HELP
  }
```

file-string

The *file-string* parameter is a character expression that specifies the pathname of a help file. If the file has a `.chm` extension (the extension for compiled Microsoft HTML Help files), the Microsoft HTML Help viewer is launched. If the file has a `.hlp` file extension, the Microsoft Windows Help viewer is launched.

`WINDOW-NAME` *`window-name`*

This option is supported for Windows Help (`.hlp` files) only.

The *`window-name`* parameter is a character expression that evaluates to the primary or secondary window name as defined in the [WINDOWS] section of the help project file. If the window name is omitted, or if “main” is specified, the primary help window is used.

`CONTENTS`

Supported only for backward compatibility.

For HTML Help, this option displays the Microsoft HTML Help viewer with the default topic in the content pane. Use the HELP-TOPIC option to specify the topic to display.

For Windows Help, this option displays the help topic defined as the contents in the [OPTIONS] section of the help project file.

`CONTEXT` *`int-expr`*

Displays the help topic that the context number identifies. You define context numbers in the [MAP] section of the help project file.

The *`int-expr`* parameter is the context number for the help topic.

`HELP-TOPIC` *`string`*

This option is supported for HTML Help (`.chm` files) only.

Displays a help topic in the content pane of the Microsoft HTML Help viewer.

The *`string`* parameter is a character expression that indicates the topic (`.htm`/`.html` file) within the compiled Microsoft HTML Help (`.chm`) file to display.

`KEY` *`string`*

For HTML Help, this option displays the topic matching the string found in the keyword index. Use semicolons in the *`string`* parameter to delimit multiple keywords. If no match is found, Microsoft HTML Help displays the help viewer with the Index tab on top.

For Windows Help, this option displays the help topic matching the string found in the index keyword list. If there is more than one match, it displays the first topic containing the keyword. If there is no match or the string is omitted, a message is displayed indicating that the keyword is invalid. The *`string`* parameter is a character expression that evaluates to a keyword for the desired help topic.

`ALTERNATE-KEY` *`string`*

This option is supported for HTML Help (`.chm` files) only. For Windows Help (`.hlp` files), see the MULTIPLE–KEY option.

Displays a help topic matching the *`string`* found in the alternate keyword (Alink) index.

The *`string`* parameter is a character expression that evaluates to a keyword in the alternate keyword index.

`POSITION X` *`x`* `Y` *`y`* `WIDTH` *`dx`* `HEIGHT` *`dy`*

Positions an existing (already opened) help window as specified.

The *`x`* parameter is an integer expression that specifies the x coordinate for the help window.

The *`y`* parameter is an integer expression that specifies the y coordinate for the help window.

The *`dx`* parameter is an integer expression that specifies the width of the help window.

The *`dy`* parameter is an integer expression that specifies the height of the help window.

`POSITION MAXIMIZE`

Maximizes an existing (already opened) help window.

`QUIT`

Informs the help application that help is no longer required. If no other applications are using help, the operating system closes the help application.

`SET-CONTENTS` *int-expr*

Supported only for backward compatibility. This option is supported for Windows Help (`.hlp` files) only.

Dynamically re-maps the contents help topic from what is defined in the [OPTIONS] section of the help project file. When a CONTENTS call is made, the new contents help topic is displayed.

The *int-expr* parameter is the context number for the new contents help topic.

`CONTEXT-POPUP` *int-expr*

This option is supported for Windows Help (`.hlp` files) only.

Displays the help topic in a pop-up window that the context number identifies. You define context numbers in the [MAP] section of the help project file. If a non-scrolling region exists in a help topic, only that region displays when you use the CONTEXT-POPUP option to display the topic.

The *int-expr* parameter is the context number for the help topic.

`PARTIAL-KEY` *string*

This option is supported for Windows Help (`.hlp` files) only.

Displays the help topic matching the string found in the keyword list. In Windows, if there is more than one match, no match, or if the string is omitted, it displays the Help Topics: Window Help Topics dialog box with the Index tab on top.

The *string* parameter is a character expression that evaluates to a partial key for the desired help topic.

`MULTIPLE-KEY` *char* `TEXT` *string*

This option is supported for Windows Help (`.hlp` files) only. For HTML Help, see the ALTERNATE-KEY option.

Displays the help topic matching a keyword from an alternate keyword table.

The *char* parameter is a character expression that evaluates to the single character keyword table identifier for the required table.

The *string* parameter is a character expression that evaluates to the keyword that is located in the keyword table.

`COMMAND` *`string`*

This option is supported for Windows Help (`.hlp` files) only.

Executes a help macro.

The *`string`* parameter is a character expression that evaluates to the help macro to execute.

`FINDER`

This option is supported for Windows Help (`.hlp` files) only.

Displays the Help Topics: Windows Help Topics dialog box, which contains an Index tab, a Find tab, and optionally a Contents tab, with the most recently used tab displayed on top.

If a Contents tab file (`.CNT` file) is present when you initially call the Help Topics: Windows Help dialog box, then the Content tab displays on top. However, if a `.CNT` file is not present, then the dialog box displays with the Index tab on top; the Contents tab is not available.

`FORCE-FILE`

This option is supported for Windows Help (`.hlp` files) only.

Ensures that the correct help file is open and displayed.

`HELP`

This option is supported for Windows Help (`.hlp` files) only.

Displays the contents of the Progress Help-on-Help file. In Windows, HELP displays the Help Topics: Windows Help Topics dialog box.

Example

The following example demonstrates several features of the SYSTEM-HELP statement with the Procedure Editor help file (`editeng.chm`). The user can select a button to demonstrate each of the following SYSTEM-HELP options: CONTEXT, KEY, ALTERNATE-KEY, POSITION, POSITION-MAXIMIZE, and QUIT.

To execute this procedure, first copy the `editeng.chm` file from `DLC\prohelp` to your current working directory (by default, `C:\OpenEdge\WRK`). Then open and run `r-syshlpchm.p` in the Procedure Editor.

r-syshlpchm.p *(1 of 2)*

```
/* r-syshlpchm.p */

DEFINE VAR helpfile as CHAR.

DEFINE BUTTON b_context LABEL "CONTEXT Call".
DEFINE BUTTON b_blank LABEL "KEY Call-''".
DEFINE BUTTON b_single LABEL "KEY Call-'Tools'".
DEFINE BUTTON b_full LABEL "KEY Call- Tools;Menu".
DEFINE BUTTON b_max LABEL "POSITION MAXIMIZE Call".
DEFINE BUTTON b_pos LABEL "POSITION Call".
DEFINE BUTTON b_alt LABEL "ALTERNATE-KEY Call".
DEFINE BUTTON b_quit LABEL "QUIT Call".

FORM
    skip(1) space(1) b_context space(1)
    skip(1) space(1) b_blank space(1)
    skip(1) space(1) b_single space(1)
    skip(1) space(1) b_full space(1)
    skip(1) space(1) b_max space(1)
    skip(1) space(1) b_pos space(1)
    skip(1) space(1) b_alt space(1)
    skip(1) space(1) b_quit space(1)
    skip(1) WITH FRAME x.
ENABLE ALL WITH FRAME x.

helpfile = "editeng.chm".

/* The CONTEXT call displays the help topic associated with the specified
context number of a help topic (in this case, 49256, for the Using Editor
Buffers topic). */
ON CHOOSE OF b_context IN FRAME x
DO:
    SYSTEM-HELP helpfile CONTEXT 49256.
END.

/* The KEY call brings up the topic matching the string found in the keyword
index. If the string parameter is empty or is omitted altogether, the help
viewer displays with the Index tab on top.*/
ON CHOOSE OF b_blank IN FRAME x
DO:
    SYSTEM-HELP helpfile KEY "".
END.
```

r-syshlpchm.p *(2 of 2)*

```
/* In a KEY call where the string parameter does not exactly match an index
   keyword of any help topic, the fill-in at the top of the Index tab is
   populated with the string that is passed in, and the default help topic
   is displayed. */
ON CHOOSE OF b_single IN FRAME x
DO:
    SYSTEM-HELP helpfile KEY "Tools".
END.

/* In a KEY call where the string parameter exactly
   matches a unique index keyword of a help topic, the help engine
   automatically launches a help viewer window and displays
   the matching topic. Use semicolons to delimit multiple keywords.*/
ON CHOOSE OF b_full IN FRAME x
DO:
    SYSTEM-HELP helpfile KEY "Tools;Menu".
END.

/* In an ALTERNATE-KEY call works like the KEY call but it uses the alternate
keyword (Alink) index, if one is provided. */
ON CHOOSE OF b_alt IN FRAME x
DO:
    SYSTEM-HELP helpfile ALTERNATE-KEY "Tools Menu".
END.

/* The POSITION X x Y y WIDTH dx HEIGHT dy call positions the open help window
as specified */
ON CHOOSE OF b_pos IN FRAME x
DO:
    SYSTEM-HELP helpfile POSITION X 2 Y 2 WIDTH 450 HEIGHT 450.
END.

/* The POSITION MAXIMIZE call maximizes the open help window as specified*/
ON CHOOSE OF b_max IN FRAME x
DO:
    SYSTEM-HELP helpfile POSITION MAXIMIZE.
END.

/* The QUIT call causes the help engine to terminate, unless another
   application is using help. */
ON CHOOSE OF b_quit IN FRAME x
DO:
    SYSTEM-HELP helpfile QUIT.
    RETURN.
END.

WAIT-FOR GO OF FRAME x.
```

See also FILE-INFO system handle, SEARCH function

TERMINAL function

In Windows, in graphical interfaces, TERMINAL returns WIN3. In Windows, in character interfaces, TERMINAL returns CO80, BW80, or MONO, depending on the monitor type. On UNIX, TERMINAL returns the value of the $TERM environment variable. In batch mode, TERMINAL returns a null string.

Note: Does not apply to SpeedScript programming.

Syntax

```
TERMINAL
```

Example

This one-line procedure displays the type of terminal you are using:

r-term.p

```
MESSAGE "You are currently using a" TERMINAL "terminal.".
```

See also

TERMINAL statement

TERMINAL statement

Changes terminal type during program execution. On UNIX, changes the value of the TERM environment variable.

Note: Does not apply to SpeedScript programming.

Syntax

```
TERMINAL = termid
```

termid

A terminal type string. The *termid* can also be an expression. Progress returns an error message if *termid* is not defined in the PROTERMCAP file. However, *termid* can be the word TERMINAL. The line TERMINAL=TERMINAL reinitializes the terminal.

Example

This procedure changes the terminal screen width from 80 columns to 132 columns, then back again:

r-seterm.p

```
FOR EACH customer:
  DISPLAY customer.
END.

TERMINAL = "wy60w".
OUTPUT TO TERMINAL PAGED.
FOR EACH customer:
  DISPLAY customer WITH WIDTH 132.
END.

OUTPUT CLOSE.
TERMINAL = "wy60".
DISPLAY "Back to 80 columns." WITH CENTERED.
```

Notes

- TERMINAL does not change the physical characteristics of a terminal. You must supply a valid terminal type for the existing terminal state.
- The TERMINAL statement reinitializes the function key definitions based on the specified PROTERMCAP entry. If you have used ON statements to change function key definitions, the TERMINAL statement overrides those changes.
- If a subprocedure uses a frame, the frame is composed with the width that was in effect when the subprocedure was compiled. Changing the width (terminal type) outside the scope of that procedure will not change the frame width inside the procedure unless it is recompiled.

  The following sequence of statements does not work as intended, because `subp.p` is not recompiled before its second execution:

  r-setrm1.p

```
TERMINAL = "wy60w".
RUN subp.p.
TERMINAL = "wy60".
RUN subp.p.
DISPLAY "Frame (132) too big for screen (80)" WITH CENTERED.
```

See also TERMINAL function

THIS-OBJECT system reference

A system reference for the currently running object instance within a class.

Syntax

```
THIS-OBJECT
```

Use THIS-OBJECT to pass an object reference to the currently running class object instance as a parameter or return an object reference to itself as a method return value.

You cannot use THIS-OBJECT to access the data members or methods within a class.

TIME function

Returns an integer representing the time in seconds. If the TIME function has no arguments, it returns the number of seconds since midnight. Use this function together with the STRING function to produce the time in hours, minutes, and seconds.

Syntax

```
TIME
```

Examples

In `r-time.p`, the timeleft variable is set to the result of the TIME function subtracted from the number of seconds in a day. The procedure translates this value into seconds, minutes, and hours.

r-time.p

```
DEFINE VARIABLE hour AS INTEGER.
DEFINE VARIABLE minute AS INTEGER.
DEFINE VARIABLE sec AS INTEGER.
DEFINE VARIABLE timeleft AS INTEGER.

timeleft = (24 * 60 * 60) - TIME.

/* seconds till next midnight */
sec = timeleft MOD 60.
timeleft = (timeleft - sec) / 60.

/* minutes till next midnight */
minute = timeleft MOD 60.

/* hours till next midnight */
hour = (timeleft - minute) / 60.

DISPLAY "Time to midnight:" hour  minute  sec .
```

This DISPLAY statement displays the current time.

r-time2.p

```
DISPLAY STRING(TIME,"HH:MM:SS").
```

See also

DATETIME function, DATETIME-TZ function, ETIME function, MTIME function, NOW function, TIMEZONE function

TIMEZONE function

Returns an integer representing the time zone offset from Coordinated Universal Time (UTC), in minutes. Use this function together with the STRING function to produce the time in hours, minutes, and seconds.

Note: Coordinated Universal Time (UTC) is the current universal standard for time. Local time zone values are relative to UTC. For example, Eastern Standard Time is UTC–05:00.

Syntax

```
TIMEZONE ( [ datetime-tz-expression | char-expression ] )
```

datetime-tz-expression

An expression whose value is a DATETIME-TZ.

char-expression

A character expression representing the time zone offset. The format of the expression must be +HH:MM.

If the TIMEZONE function has no arguments, it returns the client or server machine that serves as the time source for applications running during the OpenEdge session (*specified by the* TIME-SOURCE attribute).

Examples

Following is an example of using the TIMEZONE function:

```
DEF VAR v-dt-tz AS DATETIME-TZ INITIAL 2002-05-05T07:15:03.002-05:00.
DEF VAR v-tz AS INTEGER.

v-tz = TIMEZONE(“+08:00”).  /* v-tz = 480 */
v-tz = TIMEZONE (v-dt-tz).  /* v-tz = -300 */
```

See also

DATETIME function, DATETIME-TZ function, ETIME function, MTIME function, NOW function, TIME function, TIME-SOURCE attribute

TODAY function

Returns the current system date.

Syntax

```
TODAY
```

Example

This procedure prints the date in the first line at the top of each page of a report. Instead of using TODAY in the FORM statement, the procedure uses a variable to hold the date. This ensures that the same date appears on all pages of the report, even if this procedure runs through midnight.

r-today.p

```
DEFINE VARIABLE rptdate AS DATE.

OUTPUT TO PRINTER.

rptdate = TODAY.
FORM HEADER rptdate "Customer List" AT 34
      "Page" AT 66 PAGE-NUMBER FORMAT ">>>9" SKIP(2)
      WITH NO-BOX PAGE-TOP.
VIEW.
FOR EACH customer:
    DISPLAY name AT 1 address AT 31
      city + ", " + " " + state FORMAT "x(35)" at 31
      WITH NO-BOX NO-LABELS CENTERED.
END.
```

PAGE-TOP frames are re-evaluated on every new page. Therefore, if you do not use a variable for the date, a different date is displayed on the following page(s) if the report starts before midnight and ends after midnight.

See also

DATE function, DAY function, MONTH function, TIME function, WEEKDAY function, YEAR function

TO-ROWID function

Converts a string representation of a ROWID to a valid ROWID value.

Syntax

```
TO-ROWID ( rowid-string )
```

rowid-string

A string representation of a ROWID. Since ROWID values are a variable sequence of hexadecimal digits, *rowid-string* must be in the form "0x*hex-digits*", where *hex-digits* is any string of characters from 0 through 9 and A through F.

Examples

The following procedure (`r-torwid.p`) selects customer balance and credit information and displays it in a browse. You can select any number of rows to store and display more information on the selected customers.

r-torwid.p *(1 of 2)*

```
DEFINE QUERY custq FOR customer
    FIELDS (cust-num name balance credit-limit).
DEFINE BUFFER cust2 FOR customer.

DEFINE TEMP-TABLE rowtab FIELD rowchar AS CHARACTER
    INDEX rowi IS UNIQUE PRIMARY rowchar ASCENDING.

DEFINE BROWSE custb QUERY custq
    DISPLAY cust-num name balance credit-limit
WITH 10 DOWN MULTIPLE.
DEFINE VARIABLE hcustb AS WIDGET-HANDLE NO-UNDO.
DEFINE VARIABLE irow AS INTEGER NO-UNDO.
DEFINE BUTTON bstore LABEL "Store Selections".
DEFINE BUTTON bdisplay LABEL "Display Call Selections".
DEFINE BUTTON bclear LABEL "Clear Storage".
DEFINE FRAME brs-frame custb SKIP bstore bdisplay bclear.
DEFINE FRAME dsp-frame
    cust2.cust-num cust2.name cust2.phone
WITH 5 DOWN SCROLL 1.

ON CHOOSE OF bstore DO:
    DO irow = 1 TO custb:NUM-SELECTED-ROWS:
        IF custb:FETCH-SELECTED-ROW(irow) AND
           NOT CAN-FIND(rowtab WHERE STRING(ROWID(customer)) = rowchar)
        THEN DO:
            CREATE rowtab NO-ERROR.
            ASSIGN rowchar = STRING(ROWID(customer)) NO-ERROR.
        END.
    END.
END.
```

r-torwid.p *(2 of 2)*

```
ON CHOOSE OF bdisplay DO:
    CLEAR FRAME dsp-frame ALL.
    FOR EACH rowtab WITH FRAME dsp-frame:
        FIND cust2 WHERE ROWID(cust2) = TO-ROWID(rowchar).
        DISPLAY cust2.cust-num cust2.name cust2.phone.
        DOWN WITH FRAME dsp-frame.
    END.
END.

ON CHOOSE OF bclear DO:
    IF custb:DESELECT-ROWS() THEN FOR EACH rowtab:
        DELETE rowtab.
    END.
    FRAME dsp-frame:VISIBLE = FALSE.
END.

OPEN QUERY custq PRESELECT EACH customer.
ENABLE ALL WITH FRAME brs-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Thus, when you choose the bstore button, `r-torwid.p` stores the ROWID string values of all selected customer records in a temporary table. When you choose the bdisplay button, it displays the selected customer phone information in a down frame by converting each stored ROWID string to a ROWID value and finding the corresponding customer record. (The example also allows you to add selections and restart by deleting the existing selections.)

Note

Although TO-ROWID converts a properly formatted string to a ROWID value, there is no guarantee that this value corresponds to an existing record in your database.

See also

DATE function, DECIMAL function, INTEGER function, ROWID function, STRING function

TRANSACTION function

Returns a logical value that indicates whether a transaction is currently active.

Syntax

```
TRANSACTION
```

Note

The TRANSACTION function replaces `istrans.p`.,which was used in Progress Version 6 and earlier to determine whether a transaction was active.

See also

Transaction object handle

TRANSACTION-MODE AUTOMATIC statement (AppServer only)

Causes the procedure file that executes this statement to become an automatic transaction initiating procedure. This transaction initiating procedure allows you to control an automatic transaction in the context of an AppServer session.

Note: Does not apply to SpeedScript programming.

Syntax

```
TRANSACTION-MODE AUTOMATIC [ CHAINED ]
```

CHAINED

Tells the AppServer session to automatically create a new transaction every time the current transaction is either committed or rolled back.

Notes

- This statement must appear before any other executable statement in a top-level persistent procedure (transaction initiating procedure) running on the AppServer.
- You can control an automatic transaction by accessing the attributes and methods of the transaction object. You can access these attributes and methods on the transaction handle returned by the TRANSACTION attribute of any AppServer procedure handle.
- An automatic transaction remains open in an AppServer session until:
  - The current request service returns control to the client after an AppServer procedure invokes the transaction handle SET-COMMIT( ) method or SET-ROLLBACK( ) method.
  - The transaction initiating procedure is deleted from the session.
- If you specify the CHAINED option, a transaction is always active in the AppServer session until either the transaction initiating procedure is deleted or the AppServer session terminates.

- If you do not specify the CHAINED option and the transaction initiating procedure is still active, after the current transaction terminates, a client application can start a new transaction by directly calling any remote internal procedure of the transaction initiating procedure. When so executed, this remote internal procedure (which can otherwise be empty) creates a new transaction that you can control using the transaction handle.
- As long as an automatic transaction is open, you can execute any internal procedure of the current transaction initiating procedure from any other procedure running on the AppServer. However, if no automatic transaction is open, only a client application can execute such an internal procedure as a remote procedure call, which then opens an automatic transaction. If an AppServer procedure tries to execute such an internal procedure with no automatic transaction open, the procedure call returns an error.
- If a transaction is open when you delete the transaction initiating procedure, the transaction is committed or rolled back according to the value of the transaction handle DEFAULT-COMMIT attribute.

See also Transaction object handle

Trigger phrase

Defines triggers on one or more user-interface events for a single user-interface component. Use the Trigger phrase within the statement that defines or creates the associated user-interface component.

Note: Does not apply to SpeedScript programming.

Syntax

```
TRIGGERS:
  { ON event-list [ ANYWHERE ]
      {     trigger-block
          | PERSISTENT RUN procedure
              [ IN handle ]
              [ ( input-parameters ) ]
      }
  } ...
  END [ TRIGGERS ]
```

event-list

The event or events with which the trigger block is associated. To specify more than one event, separate them with commas as follows:

```
event1 , event2 ...
```

The events you can specify depend on the type of the associated widget. See the reference entry for the appropriate widget. For more information on each user interface event that Progress supports, see the "Events Reference" section on page 2171.

ANYWHERE

Specifies that the trigger is a group trigger. This means that it applies not only to the widget being defined or created, but also is a default to any widget contained within that widget. This allows you to create a default trigger for all widgets in a frame or window. You can override the group trigger by defining a trigger on the same event specifically for the widget (or by defining a group trigger on an intervening widget).

trigger-block

A sequence of 4GL statements to be executed when any of the specified events occur. The trigger block must be a single 4GL statement or a DO block.

PERSISTENT RUN *procedure* **[** IN *handle* **]** **[** (*input-parameters*) **]**

Specifies a persistent trigger; that is, a trigger that remains in effect after the current procedure terminates. A persistent trigger must be a procedure specified by *procedure*. The trigger procedure can take one or more input parameters; it cannot have any output parameters. The parameters are evaluated **when the trigger is defined**. They are **not** re-evaluated each time the trigger executes.

If you specify the IN *handle* option, *procedure* must be the name of an internal procedure defined in the external procedure specified by *handle*, where *handle* is an expression that evaluates to a valid procedure handle. The external procedure must be in scope when you run *procedure*.

Example This procedure defines triggers for two buttons:

r-trigp.p

```
DEFINE FRAME cust-frame.
DEFINE QUERY custq FOR customer.

DEFINE BUTTON nextcust LABEL "Next"
   TRIGGERS:
      ON CHOOSE
          DO:
              GET NEXT custq.
              DISPLAY customer WITH FRAME cust-frame.
          END.
   END TRIGGERS.

DEFINE BUTTON prevcust LABEL "Previous"
   TRIGGERS:
      ON CHOOSE
          DO:
              GET PREV custq.
              DISPLAY customer WITH FRAME cust-frame.
          END.
   END TRIGGERS.

OPEN QUERY custq FOR EACH customer.

GET FIRST custq.
DISPLAY customer WITH FRAME cust-frame.

ENABLE nextcust AT COLUMN 1 ROW 7 prevcust WITH FRAME cust-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Notes

- If you specify the Trigger phrase in the definition of a user-interface component, the Trigger phrase must be the last option in the component definition.
- If you specify a trigger when you define a widget then that trigger applies to every instance of that widget. For example, in `r-trigp.p`, if you enable the nextcust button in more than one frame, each of those buttons inherits the nextcust trigger.
- The input parameters for a persistent trigger are evaluated when the trigger is attached. (For the Trigger phrase, the trigger is attached when the widget is realized.) This means, for example, that you cannot pass the SELF handle as an input parameter.
- The external procedure specified by *handle* is in scope if it is the current procedure, a procedure on the call stack, or a persistent procedure.

See also CREATE widget statement, DEFINE MENU statement, ON statement

TRIGGER PROCEDURE statement

Defines a schema trigger.

Syntax

```
TRIGGER PROCEDURE FOR event OF object [ options ]
```

event

The event for which the schema trigger is being defined. The Valid events are CREATE, DELETE, FIND, WRITE, and ASSIGN.

object

The object on which the event is defined. If the event is CREATE, DELETE, FIND, or WRITE, the object must be a reference to a database table. If the event is ASSIGN, the object must be a reference to a database field qualified by a table name.

options

Optional parts of the trigger header.

Headers for CREATE, DELETE, and FIND triggers take no options. Their syntaxes are as follows:

```
TRIGGER PROCEDURE FOR CREATE OF table
```

```
TRIGGER PROCEDURE FOR DELETE OF table
```

```
TRIGGER PROCEDURE FOR FIND OF table
```

In the header for a WRITE trigger you can optionally include one or two buffer names.

```
TRIGGER PROCEDURE FOR WRITE OF table
  [ NEW [ BUFFER ] buffer-name1 ]
  [ OLD [ BUFFER ] buffer-name2 ]
```

In the header for an ASSIGN trigger, you can optionally specify one or two value holders. You can specify formatting for each as follows:

```
TRIGGER PROCEDURE FOR ASSIGN
  {     { OF table .field }
     |  { NEW [ VALUE ] value1
             { AS data-type | LIKE db-field }
        }
  }
  [ COLUMN-LABEL label ]
  [ FORMAT format-string ]
  [ INITIAL constant ]
  [ LABEL label-string ]
  [ NO-UNDO ]
  [ OLD [ VALUE ] value2
      { AS data-type | LIKE db-field }
      [ COLUMN-LABEL label ]
      [ FORMAT format-string ]
      [ INITIAL constant ]
      [ LABEL label-string ]
      [ NO-UNDO ]
  ]
```

Example

The following is a WRITE trigger for the customer table. It uses the OLD BUFFER option so that it can determine whether the cust-num value has changed. If the customer's outstanding balance exceeds its credit limit, the trigger returns the error condition (in which case the record is not updated).

r-wrcust.p

```
TRIGGER PROCEDURE FOR Write OF Customer OLD BUFFER oldCustomer.

/* Variable Definitions */
DEFINE VARIABLE i AS INTEGER INITIAL 0.
DEFINE VARIABLE Outstanding AS INTEGER INITIAL 0.

/* Check to see if the user changed the Customer Number */
IF Customer.Cust-Num <> oldCustomer.Cust-Num AND oldCustomer.Cust-Num <> 0
THEN DO:
   /* If the user changed the Customer Number, find all related orders
      and change their customer numbers.                                */

   FOR EACH order OF oldCustomer:
      Order.Cust-Num = Customer.Cust-Num.
      i = i + 1.
   END.
   IF i > 0 THEN
     MESSAGE i "orders changed to reflect the new customer number!".
END.

/* Ensure that the Credit Limit value is always Greater than the sum of
   this Customer's Outstanding Balance */
FOR EACH Order OF Customer:
   FOR EACH Order-Line OF Order:
      Outstanding = Outstanding + ( Qty * Order-Line.Price ).
   END.
END.
FOR EACH Invoice OF Customer:
   Outstanding = Outstanding + ( Amount - ( Total-Paid + Adjustment )).
END.

IF Customer.Credit-Limit < Outstanding THEN DO:
   MESSAGE "This Customer has an outstanding balance of: " Outstanding
           ".  The Credit Limit MUST exceed this amount!".
   RETURN ERROR.
END.
```

Notes

- For more information on database triggers, see *OpenEdge Development: Progress 4GL Handbook*.
- Use the Data Dictionary to associate a trigger procedure with a table or field in the database.
- Some 3GL applications execute schema triggers. Triggers might also be executed in batch mode. Therefore, you should avoid any user-interface interactions within schema trigger procedures.

See also PROCEDURE statement

TRIM function

Removes leading and trailing white space, or other specified characters, from a CHARACTER or LONGCHAR expression.

Syntax

```
TRIM ( expression [ , trim-chars ] )
```

expression

An expression (a constant, field name, variable name, or expression) whose value is a CHARACTER or LONGCHAR. If *expression* is a case-sensitive variable, Progress performs a case-sensitive trim. If *expression* is a LONGCHAR, the result is in the same code page.

trim-chars

A character expression that specifies the characters to trim from *expression*. If you do not specify *trim-chars*, the TRIM function removes spaces, tabs, line feeds, and carriage returns.

Examples

The following procedure displays a menu that you can use to display customer and order information. The option numbers are displayed with leading spaces. The TRIM function removes the leading white space so the menu selection can be easily evaluated.

r-trim.p

```
DEFINE VARIABLE menu AS CHARACTER EXTENT 3.
DO WHILE TRUE:
  DISPLAY
      "    1.    Display Customer Data" @ menu[1] SKIP
      "    2.    Order Data" @ menu[2] SKIP
      "    3.    Exit" @ menu[3] SkIP
  WITH FRAME choices NO-LABELS.
  CHOOSE FIELD menu AUTO-RETURN WITH FRAME choices
  TITLE "Demonstration Menu" CENTERED ROW 10.
  HIDE FRAME choices.
  IF TRIM(FRAME-VALUE) BEGINS "1" THEN RUN r-dblnkc.p.
  IF TRIM(FRAME-VALUE) BEGINS "2" THEN RUN r-dblnko.p
  IF TRIM(FRAME-VALUE) BEGINS "3" THEN LEAVE.
END.
```

The following example reads a text file and breaks it into words. It assumes that all words are separated by at least one space character. It uses the TRIM function with one parameter to remove white space from the ends of each input line. It then uses the TRIM function with two parameters to remove any punctuation characters from each word.

r-trim2.p

```
DEFINE VARIABLE infile AS CHARACTER FORMAT "x(60)" LABEL "Input File".
DEFINE VARIABLE intext AS CHARACTER.
DEFINE VARIABLE next-space AS INTEGER.
DEFINE VARIABLE word AS CHARACTER FORMAT "x(32)" LABEL "Words".

/* Get the name of a text file and set input to that file. */.
SET infile.
INPUT FROM VALUE(infile).

DO WHILE TRUE:
  /* Read the next line from the file. */
  IMPORT UNFORMATTED intext.
  intext = TRIM(intext).

  DO WHILE TRUE:
    /* Find the next space character.  If none found, find
      the end of string.                                    */
    next-space = INDEX(intext, " ").
    IF next-space = 0
    THEN next-space = LENGTH(intext) + 1.

    /* If the string contains no (more) words, then read the
      next line.                                             */
    IF next-space = 1
    THEN LEAVE.

    /* Pull the first word off the string.
      Remove any punctuation characters around it. */
    word = SUBSTRING(intext, 1, next-space - 1).
    word = TRIM(word, ",.;:!? ~"~ '[]()").
    intext = TRIM(SUBSTRING(intext, next-space + 1)).

    /* Display the word. */
    DISPLAY word WITH DOWN FRAME x.
    DOWN WITH FRAME x.
  END.
END.
```

Notes

- The TRIM function is double-byte enabled. The specified *expression* and *trim-chars* arguments can contain double-byte characters. TRIM does not remove double-byte space characters by default.
- A character string displays with the default format of x(8), unless you specify a format or use a statement such as DISPLAY @ *literal*.
- You can use the DEBLANK option of the Format phrase to remove leading spaces for fields in the input buffer.
- If *expression* is a case-sensitive field or variable, then *trim-chars* is also case sensitive. Otherwise, *trim-chars* is not case sensitive.

See also LEFT-TRIM function, RIGHT-TRIM function

TRUNCATE function

Truncates a decimal expression to a specified number of decimal places, returning a decimal value.

Syntax

```
TRUNCATE ( expression , decimal-places )
```

expression

A decimal expression that you want to truncate.

decimal-places

A non-negative integer expression that indicates the number of decimal places for a truncated *expression*.

Example

This procedure doubles each customer's credit-limit and then truncates that value before rounding it to the nearest $1000:

r-trunc.p

```
FOR EACH customer:
  FORM cust-num name credit-limit
    new-max LIKE credit-limit LABEL "New Credit limit".
  DISPLAY cust-num name credit-limit.
  credit-limit = TRUNCATE( (credit-limit * 2) / 1000 ,0) * 1000.
  IF credit-limit < 15000 THEN credit-limit = 15000.
  DISPLAY credit-limit @ new-max.
END.
```

Note

You can use the TRUNCATE function to treat division as integer division. For example, i = TRUNCATE (x / y, 0).

See also

ROUND function

Type-name syntax

Specify a user-defined type name for a class, a super class, or an interface as a character string using the following syntax:

```
[package.]class-name
```

package

A period-separated list of components that, along with the class name, uniquely identify the class, super class, or interface. The components are based on a valid directory pathname relative to PROPATH, where each component maps to a directory level in the path and each slash separator in the path is replaced with a period. If specified, the relative path of the class definition file represented by *package* must remain constant between compile time and run time.

Note: Do not place a class definition file in a directory whose name contains a period (.) character; Progress interprets the component after the period as another directory level and will not find the referenced class definition file.

class-name

The class name. This name must match the name of a class definition file (excluding the .cls or .r extension) located in the relative path represented by *package*, if specified.

This name must begin with an alphabetic character and it cannot contain a period.

Do not name a class type using a Progress reserved keyword or a built-in Progress data type name, such as INTEGER. For a list of Progress keywords, see the "Keyword Index" section on page 2209.

If *package* or *class-name* contains embedded spaces, you must enclose the entire type name in quotes.

Example

If your PROPATH is "C:/myfiles", and your class definition file name is "C:/myfiles/acme/myObjs/CustObjs.cls", then Progress requires *package* to be "acme.myObjs." and *class-name* to be "CustObjs".

TYPE-OF function

Verifies that the object instance to which the specified object reference applies is an instance of the specified class type, inherits from the specified super class, or implements the specified interface. If the object reference represents an object instance of the specified type, the function returns TRUE. Otherwise, it returns FALSE.

Syntax

```
TYPE-OF ( object-reference, type-name ).
```

object-reference

An object reference for a user-defined class object instance.

type-name

A character string that specifies the type name of a class, a super class, or an interface. Specify a type name using the *package.class-name* syntax as described in the Type-name syntax reference entry in this book.

UNDERLINE statement

Underlines a field or variable, using the next display line for the underline.

Note: Does not apply to SpeedScript programming.

Syntax

```
UNDERLINE [ STREAM stream ] field ... [ frame-phrase ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

field

Represents the name of the field or variable you want to underline.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

This procedure produces a report of customer records, categorized by state. When the last customer for a certain state has been displayed (determined by the LAST-OF function), the UNDERLINE statement underlines the state field.

r-under1.p

```
FOR EACH customer BREAK BY state WITH USE-TEXT:
  DISPLAY state cust-num name.
  IF LAST-OF(state) THEN UNDERLINE state.
END.
```

Notes

- Use The UNDERLINE statement to highlight fields or to underline accumulated values that you calculated using functions other than the automatic aggregate functions supplied with Progress.
- When determining the position within a DOWN frame, the DOWN statement and the UP statement count the line used by an underline.
- Even if the layout of a DOWN frame takes multiple screen lines, the underline takes just one line on the screen.
- For a 1 DOWN frame or single frame, the UNDERLINE does not appear. Instead, Progress clears the frame.

See also DEFINE STREAM statement, Frame phrase

UNDO statement

Backs out all modifications to fields and variables made during the current iteration of a block, and indicates what action to take next.

Syntax

```
UNDO
  [ label ]
  [     , LEAVE [ label2 ]
      | , NEXT [ label2 ]
      | , RETRY [ label1 ]
      | , RETURN [ ERROR | NO-APPLY ] [ return-value  ]
  ]
```

label1

The name of the block whose processing you want to undo. If you do not name a block with *label1*, UNDO undoes the processing of the closest transaction or subtransaction block. In determining the closest transaction or subtransaction block, Progress disregards DO ON ENDKEY blocks that do not have the ON ERROR or TRANSACTION option.

LEAVE *label2*

Indicates that after undoing the processing of a block, Progress leaves the block you name with *label2*. If you do not name a block with the LEAVE option, Progress leaves the block that was undone. After leaving a block, Progress continues on with any remaining processing in a procedure.

NEXT *label2*

Indicates that after undoing the processing of a block, Progress does the next iteration of the block you name with *label2*. If you do not name a block, Progress does the next iteration of the block that was undone.

RETRY *label1*

Indicates that after undoing the processing of a block, Progress repeats the same iteration of the block you name with *label1*. If you name a block with *label1* it must be the name of the block that was undone.

RETRY is the default processing if you do not use LEAVE, NEXT, RETRY, or RETURN. When a block is retried, any frames scoped to that block are not advanced or cleared. For more information, see *OpenEdge Development: Progress 4GL Handbook*.

RETURN [ERROR | NO-APPLY]

Returns to the calling procedure or to the Progress Editor (if there was no calling procedure). You can specify NO-APPLY within a user-interface trigger to prevent Progress from doing anymore handling for the event. In other contexts, you can use the ERROR option to raise the ERROR condition in the caller.

return-value

You can use *return-value* to return a value to the caller.

Example

The r-undo.p procedure prompts you for the initials of a sales representative. If the initials match those of an existing sales representative, the procedure displays that sales representative's record. Otherwise, it prompts you to add another sales representative with the initials you supplied. If you enter no, the UNDO statement undoes the work you have done since the start of the REPEAT block and lets you enter another set of initials.

r-undo.p

```
DEFINE VARIABLE ans AS LOGICAL.
REPEAT FOR salesrep WITH ROW 7 1 COLUMN 1 DOWN CENTERED
    ON ENDKEY UNDO, LEAVE:
  PROMPT-FOR sales-rep.
  FIND salesrep USING sales-rep NO-ERROR.
  IF NOT AVAILABLE salesrep
  THEN DO:
    ans = yes.
    MESSAGE "Salesrep record does not exist.".
    MESSAGE "Do you want to add a salesrep?" UPDATE ans.
    IF ans THEN DO:
      CREATE salesrep.
      ASSIGN sales-rep.
      UPDATE rep-name region month-quota.
    END.
    ELSE UNDO, RETRY.
  END.
  ELSE DISPLAY salesrep.
END.
```

Notes

- You can also specify UNDO processing for a block by using the ON ERROR and ON ENDKEY phrases with a block statement.
- An UNDO statement that specifies a block that encompasses the current system transaction block has no effect on changes made prior to the start of the system transaction. This includes changes made to variables prior to the beginning of the system transaction.
- If nothing changes during a RETRY of a block, then the RETRY is treated as a NEXT or a LEAVE. This default action provides protection against infinite loops.
- For more information on the UNDO statement, see *OpenEdge Development: Progress 4GL Handbook*.

See also ON ENDKEY phrase, ON ERROR phrase, RETRY function

UNIX statement (UNIX only)

Runs a program, UNIX command ,or UNIX script, or starts a UNIX interactive shell to allow interactive processing of UNIX commands.

Syntax

```
UNIX
  [SILENT]
  [ command-token | VALUE ( expression ) ] ...
```

SILENT

After processing a UNIX statement, the Progress shell pauses and prompts you to press **SPACEBAR** to continue. You can use the SILENT option to eliminate this pause. Use this option only if you are sure that the UNIX program, command, or batch file does not generate any output to the screen.

command-token | VALUE (*expression*)

One or more command (*command-token*) words and symbols that you want to pass the UNIX operating system to execute. The VALUE option generates the command tokens included in *expression*, a character string expression. The specified combination of *command-token* and VALUE (*expression*) options can form any legal combination of commands and command options permitted by UNIX, including programs, built-in commands, and scripts. If you do not use any of these options, the UNIX statement invokes the UNIX shell and remains there until you press **CTRL+D** or the EOF character set by the UNIX `stty` command.

Examples

On UNIX, procedure `r-unix.p` starts a shell and in it runs the UNIX “ls” command. In Windows, this procedure starts a command processor and in it runs the DOS “dir” command:

r-unix.p

```
IF OPSYS = "UNIX" THEN UNIX ls.
ELSE IF OPSYS = "WIN32" THEN DOS dir.
ELSE DISPLAY OPSYS "is an unsupported operating system".
```

In r-unx.p, if you type an **L**, Progress runs the DOS dir command or the UNIX ls command. If you enter a procedure name that is stored in the proc variable, the RUN statement then runs the procedure.

r-unx.p

```
DEFINE VARIABLE proc AS CHARACTER FORMAT "x(40)".

REPEAT:
  DISPLAY "Enter L to list your files"
    WITH ROW 5 CENTERED FRAME a.
  SET proc LABEL "Enter a valid Procedure Name to run"
    WITH ROW 9 CENTERED FRAME b.
  IF proc = "L" THEN
    IF OPSYS = "UNIX" THEN UNIX ls.
    ELSE IF OPSYS = "WIN32" then DOS dir.
    ELSE display "Operating system" OPSYS "is not supported".

  ELSE DO:
    HIDE FRAME a.
    HIDE FRAME b.
    RUN VALUE(proc).
   END.
END.
```

Notes

- If you are using Windows and you use the UNIX statement in a procedure, that procedure will compile. The procedure will run as long as flow of control does not pass through the UNIX statement.
- This command does not exit to UNIX and return. It creates a shell within Progress to execute the command. Thus, you cannot use the UNIX statement as a substitute for the QUIT statement.
- When you use the UNIX `cp` command as a Progress statement, Progress assumes that a period (.) indicates the end of the statement. This causes the `cp` command to display a message stating that it requires two arguments. For example, Progress uses the period as the end of the statement indicator:

```
UNIX cp usr/myfile
```

To use the period as part of a UNIX command, enclose the command in quotation marks. For example:

```
UNIX "cp usr/myfile."
```

See also DOS statement, OPSYS function, OS-COMMAND statement

UNLOAD statement (Windows only)

Unloads a set of environment specifications from the current environment, which might be the registry or an initialization file.

Note: Does not apply to SpeedScript programming.

Syntax

```
UNLOAD environment [ NO-ERROR ]
```

environment

A character expression that evaluates to the name of an environment that a prior LOAD statement specified.

NO-ERROR

Directs Progress to suppress any errors that occur in the attempt to unload the environment specifications. After the UNLOAD statement completes, you can check the ERROR-STATUS system handle for information on suppressed errors.

Notes

- An application cannot UNLOAD a set of environment specifications until it terminates all windows that use those specifications.
- If you UNLOAD the current environment, the default environment becomes the current environment. To define a new current environment, use the USE statement.
- Use the UNLOAD statement to clean up memory in applications, such as the User Interface Builder, that build and run other applications.

See also

LOAD statement, USE statement

UNSUBSCRIBE statement

Cancels a subscription to a Progress named event. Specifically, the UNSUBSCRIBE statement cancels one or more subscriptions to one or more named events.

Note: Progress named events are completely different from the key function, mouse, widget, and direct manipulation events, which are described in the "Events Reference" section on page 2171. For more information on Progress named events, see *OpenEdge Development: Progress 4GL Handbook*.

Syntax

```
UNSUBSCRIBE [ PROCEDURE subscriber-handle ]
  [ TO ] { event-name | ALL } [ IN publisher-handle ]
```

PROCEDURE *subscriber-handle*

A procedure handle representing the subscriber to a named event.

The PROCEDURE option lets one procedure cancel a subscription on behalf of another. For example, if you want procedure A to cancel a subscription on behalf of procedure B, set *subscriber-handle* to the procedure handle of B.

If the PROCEDURE option does not appear, Progress assumes that the subscriber is THIS-PROCEDURE, the procedure that contains the UNSUBSCRIBE statement,

event-name

A quoted string or character expression representing the name of a named event.

ALL

Cancels all subscriptions.

`IN` *`publisher-handle`*

A procedure handle representing the publisher of a named event.

If the IN option appears, Progress cancels subscriptions to named events published by *`publisher-handle`*-specifically, either all subscriptions (if the ALL option appears), or only subscriptions to *`event-name`* (if *`event-name`* appears).

If the IN option does not appear, Progress cancels subscriptions regardless of the publisher-specifically, either all subscriptions (if the ALL option appears), or only subscriptions to *`event-name`* (if *`event-name`* appears).

Example

For an example, see the reference entry for the PUBLISH statement in this book.

Notes

- When Progress executes an UNSUBSCRIBE statement, it cancels a subscription when it finds a match. A match means that the SUBSCRIBE and UNSUBSCRIBE event names match, and that one of the following is true:
  - The subscription was created using SUBSCRIBE IN, cancelled using UNSUBSCRIBE IN, and the publisher and subscriber handles in the SUBSCRIBE and UNSUBSCRIBE statements match.
  - The subscription was created using SUBSCRIBE ANYWHERE, and cancelled using UNSUBSCRIBE without the IN option.
- Progress executes the UNSUBSCRIBE statement with an implicit NO-ERROR option. That is, if Progress cannot find a match, it does not report an error. To find out what errors, if any, occurred, use the ERROR-STATUS system handle.
- If you create a subscription using SUBSCRIBE ANYWHERE, you cannot cancel the subscription using UNSUBSCRIBE IN.

See also

PUBLISH statement, PUBLISHED-EVENTS attribute, SUBSCRIBE statement

UP statement

Positions the cursor on a new line in a down or multi-line frame.

When the block specifying the down frame iterates, Progress automatically advances one frame line. Use the UP statement if you want to move to a different display line at any time.

For more information on down frames, see the DOWN option of the Frame phrase.

Note: Does not apply to SpeedScript programming.

Syntax

```
UP [ STREAM stream ] [ expression ] [ frame-phrase ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream. See the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces* for more information on streams.

expression

Represents the number of occurrences of data in the frame that you want to move up. UP is the same as UP 1, except that nothing happens until the next data handling statement affects the screen. Several UP statements in a row with no intervening displays are treated like a single UP 1. UP 0 does nothing. If *expression* is negative, the result is the same as a DOWN *expression*.

frame-phrase

Specifies the overall layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

Example

This procedure starts at the bottom of the screen and displays all the customer database records. The default frame for the FOR EACH block is a down frame. The DISPLAY statement uses that frame. Therefore, Progress automatically advances down the screen one line after each iteration. You must use an UP 2 rather than an UP 1 because there is an automatic DOWN 1 performed on the display frame at the end of each iteration of the FOR EACH block.

r-up.p

```
FOR EACH customer:
  UP 2.
  DISPLAY cust-num name city.
END.
```

Notes

- When a frame is a down frame, Progress automatically advances to the next frame line on each iteration of the block that it is scoped to, whether or not you use the DOWN statement. If you do not want Progress to do this automatic advancing, name the frame outside of the current block. For more information on frames, see *OpenEdge Development: Progress 4GL Handbook*.
- When Progress reaches the top frame line and then encounters an UP statement, it clears the frame and starts at the bottom line of the frame. However, if you use SCROLL, Progress moves everything in the frame down one row.

See also

DEFINE STREAM statement, DOWN statement, Frame phrase, SCROLL statement

UPDATE statement

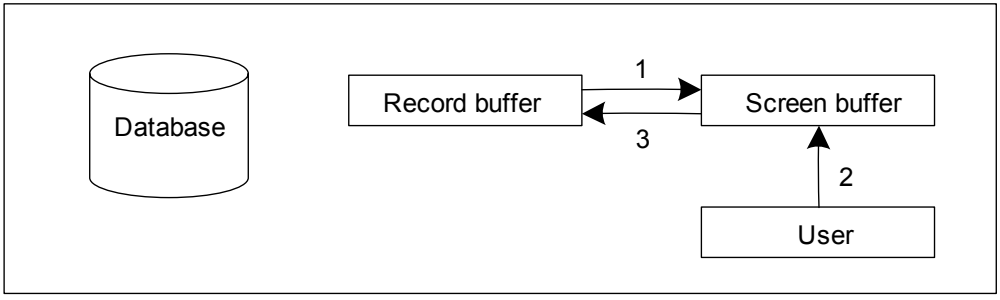
Displays fields or variables, requests input, and then puts the input data in both the screen buffer and in the specified fields or variables.

The UPDATE statement is a combination of the following statements:

- **DISPLAY** — Moves the values of fields or variables into the screen buffer and displays them.
- **PROMPT-FOR** — Prompts the user for data and puts that data into the screen buffer.
- **ASSIGN** — Moves data from the screen buffer to the record buffer.

Note: Does not apply to SpeedScript programming.

Data movement



Syntax

```
UPDATE
  [ UNLESS-HIDDEN ]
  [   field [ format-phrase ] [ WHEN expression ]
    | TEXT ( field [ format-phrase ] ... )
    | field = expression
    | constant [ AT n | TO n ]
    | ^
    | SPACE [ ( n ) ]
    | SKIP [ ( n )]
  ] ...
  [ GO-ON ( key-label ... ) ]
  [ frame-phrase ]
  [ editing-phrase ]
  [ NO-ERROR ]
```

```
UPDATE record
  [ EXCEPT field ... ]
  [ frame-phrase ]
  [ NO-ERROR ]
```

UNLESS-HIDDEN

Restricts UPDATE to fields whose HIDDEN attribute is FALSE.

field

Represents the name of the field or variable whose value you want to display, change, and store in the screen and record buffers.

In array fields, array elements with constant subscripts are handled as any other field. Array fields with no subscripts are expanded as though you entered the implicit elements. See the DISPLAY statement reference entry for information on how array fields with expressions as subscripts are handled.

You can supply values for array elements in the UPDATE statement as follows:

```
UPDATE x[1] = "x".
```

This statement assigns the letter x to the first element of array x. If you do not include an array subscript, Progress assigns the value to all elements of the array.

```
UPDATE X = "X".
```

This statement assigns the letter x to all elements of the array x.

format-phrase

Specifies one or more frame attributes for a field, variable, or expression. For more information on *format-phrase*, see the Format phrase reference entry.

WHEN *expression*

Updates the field only when *expression* has a value of TRUE. The *expression* is a field name, variable name, or expression whose value is logical.

TEXT

Defines a group of character fields or variables (including array elements) to use automatic word wrap. The TEXT option works only with character fields that are Progress default FILL-IN widgets (not specified with the FILL-IN NATIVE option). When you insert data in the middle of a TEXT field, Progress wraps data that follows into the next TEXT field, if necessary. If you delete data from the middle of a TEXT field, Progress wraps data that follows to the empty area. If you enter more characters than the format for the field allows, Progress discards the extra characters. The character fields formats must be in the x(n) format. A blank in the first column of a line marks the beginning of a paragraph. Lines within a paragraph are treated as a group and will not wrap into other paragraphs.

Table 50 lists the keys you can use within a TEXT field and their actions.

Table 50: Key actions in a TEXT() field

Key	Action
APPEND-LINE	Combines the line the cursor is in with the next line.
BACK-TAB	Moves the cursor to the previous TEXT field.
BREAK-LINE	Breaks the current line into two lines beginning with the character the cursor is in.
BACKSPACE	Moves the cursor one position to the left and deletes the character at that position. If the cursor is at the beginning of a line, **BACKSPACE** moves the cursor to the end of the previous line.
CLEAR	Clears the current field and all fields in the TEXT group that follow.
DELETE-LINE	Deletes the line the cursor is in.
NEW-LINE	Inserts a blank line below the line the cursor is in.
RECALL	Clears fields in the TEXT group and returns initial data values for the group.
RETURN	In overstrike mode, moves to the next field in the TEXT group on the screen. In insert mode, the line breaks at the cursor and the cursor is positioned at the beginning of the new line.
TAB	Moves to the field after the TEXT group on the screen. If there is no other field, the cursor moves to the beginning of the TEXT group.

In this procedure, the s-com field is a TEXT field. Run the procedure and enter text in the field to see how the TEXT option works:

r-text.p

```
DEFINE VARIABLE s-com AS CHARACTER FORMAT "x(40)" EXTENT 5.

FORM "Shipped   :" order.ship-date AT 13 SKIP
     "Misc Info :" order.instructions AT 13 SKIP(1)
     "Order Comments :" s-com AT 1
WITH FRAME o-com CENTERED NO-LABELS TITLE "Shipping Information".

FOR EACH customer, EACH order OF customer:
    DISPLAY cust.cust-num cust.name order.order-num order.order-date
            order.promise-date WITH FRAME order-hdr CENTERED.
    UPDATE ship-date instructions TEXT(s-com) WITH FRAME o-com.
    s-com = "".
END.
```

field = expression

Indicates that the value of *field* is determined by evaluating the expression rather than having it entered on the screen or from a file. In effect, an assignment statement is embedded in the UPDATE statement.

constant AT *n*

Represents a constant value that you want to display in the frame. The *n* is the column in which you want to start the display.

constant TO *n*

Represents a constant value that you want to display in the frame. The *n* is the column in which you want to end the display.

^

Tells Progress to ignore an input field when input is being read from a file. Also, the following statement reads a line from an input file and ignore that line:

```
UPDATE^
```

SPACE [(*n*)]

Identifies the number (*n*) of blank spaces to insert after the expression displays. The *n* can be 0. If the number of spaces you specify is more than the spaces left on the current line of the frame, Progress starts a new line and discards any extra spaces. If you do not use this option or do not use *n*, Progress inserts one space between items in the frame.

SKIP [(*n*)]

Identifies the number (*n*) of blank lines to insert after the expression is displayed. The *n* can be 0. If you do not use this option, Progress does not skip a line between expressions unless they do not fit on one line. If you use the SKIP option, but do not specify *n* or if *n* is 0, Progress starts a new line unless it is already at the beginning of a new line.

GO-ON (*keylabel* . . .)

Tells Progress to take the GO action when the user presses any of the keys listed. You list keys in addition to keys that perform the GO action by default or because of ON statements. For example, if you want Progress to execute the GO action when the user presses **F1**, use the statement GO-ON(**F1**). If you list more than one key, separate them with spaces, not commas.

Note that the GO-ON option is valid if you specify a list of fields in the UPDATE statement, but is invalid if you specify a record.

frame-phrase

Specifies the layout and processing properties of a frame. For more information on *frame-phrase*, see the Frame phrase reference entry.

editing-phrase

Supported only for backward compatibility.

Identifies processing to take place as each keystroke is entered. This is the syntax for *editing-phrase*:

```
[ LABEL : ] EDITING : statement ... END
```

For more information on *editing-phrase*, see the EDITING phrase reference entry.

NO-ERROR

Specifies that any errors that occur in the attempt to update a record are suppressed. After the UPDATE statement completes, you can check the ERROR-STATUS system handle for information on any errors that occurred.

record

Specifies the name of a record buffer. All of the fields in the record are processed as if you updated each of them individually.

To update a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

EXCEPT *field*

Affects all fields except those fields listed in the EXCEPT phrase; they are omitted from the update list.

Examples

The following procedure lets you update the name, address, city, state, and country for each customer record in the database:

r-updat.p

```
FOR EACH customer:
  UPDATE name address city state country.
END.
```

The r-updat2.p procedure reads each customer record and lets you update the name and credit-limit fields. The VALIDATE option on the first UPDATE statement ensures that you enter a credit-limit value that is less than 500000. The HELP option displays a message to that effect.

r-updat2.p

```
FOR EACH customer:
  UPDATE customer.name
    credit-limit VALIDATE(credit-limit < 500000, "Too high")
      HELP "Enter credit-limit < 500000".
    FOR EACH order OF customer:
      DISPLAY order-num.
      UPDATE promise-date ship-date VALIDATE(ship-date > today,
          "Ship date must be later than today").
    END.
END.
```

The second FOR EACH block reads every order belonging to the customer, displays the order-num field, and lets you update the promise-date and ship-date fields. The VALIDATE option ensures that you enter a ship date value that is after today's date.

This procedure requests a customer number and then lets you update information for that customer record. The frame phrase WITH 1 COLUMN 1 DOWN tells Progress to display the fields in a single column on the screen (rather than in a row across the screen) and to display only one customer record on the screen at a time.

r-updat3.p

```
REPEAT:
  PROMPT-FOR customer.cust-num.
  FIND customer USING cust-num.
  UPDATE name address city customer.state country WITH 1 COLUMN 1 DOWN.
END.
```

Notes

- If any *field* is a field in a database record, the UPDATE statement upgrades the record lock condition to EXCLUSIVE-LOCK before updating the record.
- If any *field* is part of a record retrieved with a field list, the UPDATE statement rereads the complete record before updating it. If any *field* is not part of the field list (or related fields) fetched with the record, or if *record* includes such unfetched fields, Progress raises the ERROR condition **before** the UPDATE statement accepts input. This is because the UPDATE attempts to display the fields before it rereads the record.
- If an error occurs during UPDATE statement input (for example, the user enters a duplicate index value for a unique index), Progress retries the data entry part of the statement and does not do the error processing associated with the block that contains the statement.

- The UPDATE statement is **not** equivalent to a combination of the DISPLAY and SET statements.

```
REPEAT:
  PROMPT-FOR customer.cust-num.
  FIND customer USING cust-num.
  UPDATE credit-limit.
END.
```

The previous procedure is approximately equivalent to the following procedure:

```
REPEAT:
  PROMPT-FOR customer.cust-num.
  FIND customer USING cust-num.
  DISPLAY credit-limit.
  DO ON ERROR UNDO, RETRY:
    SET credit-limit.
  END.
END.
```

If an error occurs during an UPDATE statement, the statement is retried until the error is corrected. If this happens during a SET statement, an entire block is retried.

- If you receive input from a device other than the terminal, and the number of characters read by the UPDATE statement for a particular field or variable exceeds the display format for that field or variable, Progress returns an error. However, if you are setting a logical field that has a y/n format and the data file contains a value of YES or NO, Progress converts that value to “y” or “n”.

- If you use a single qualified identifier with the UPDATE statement, the compiler first interprets the reference as *dbname.tablename*. If the compiler cannot resolve the reference as *dbname.tablename*, it tries to resolve it as *tablename.fieldname*.

- When updating fields, you must use table names that are different from field names to avoid ambiguous references. See the Record phrase reference entry for more information.

- The UPDATE statement causes ASSIGN and WRITE events to occur and all related database ASSIGN and WRITE triggers to execute. The ASSIGN triggers execute before the WRITE triggers and after the field is actually updated. The WRITE triggers only execute if the ASSIGN triggers do not return an error. If an ASSIGN trigger fails, the database update is undone. This means that all database changes are backed out. If the UPDATE statement occurs within a transaction, any changes to variables, worktable fields, and temporary table fields are also undone unless the variable or field is defined with the NO-UNDO option. Likewise, if a WRITE trigger fails, the UPDATE statement is undone. See *OpenEdge Development: Progress 4GL Handbook* for more information on database triggers.

- In Progress Version 7 and above, when you execute UPDATE with a specific or implied GO-ON(*key1abe1*) from a called program, Progress generates an error message (4123). This is due to an incompatibility in focus. The workaround is to add a VIEW FRAME statement after the call to the subprocedure such that the VIEW FRAME is the first statement executed on return from the called procedure.

See also ASSIGN statement, DISPLAY statement, EDITING phrase, Format phrase, Frame phrase, PROMPT-FOR statement

USE statement (Windows only)

Specifies environment defaults that apply to subsequent windows that the application creates. The defaults might reside in the registry or in an initialization file. The defaults can involve colors, fonts, environment variables, etc. You must specify a default in a LOAD statement before you specify it in a USE statement.

Note: Does not apply to SpeedScript programming.

Syntax

```
USE environment [ NO-ERROR ]
```

environment

A CHARACTER expression that evaluates to the name of a current environment. If *environment* is non-null, it must have appeared in a prior LOAD statement. If *environment* is the null string (""), the default environment becomes the current environment.

NO-ERROR

Directs Progress to suppress any errors that occur in the attempt use the environment file specifications. After the USE statement completes, you can check the ERROR-STATUS system handle for information on suppressed errors.

Example

This procedure loads two files, env1.ini and env2.ini, each of which contains a font definition for font0. The program displays a character string in the Progress default window using the definition for font0 from env1.ini. It then creates a new window and displays the same character string using the definition for font0 from env2.ini. Note that the procedure creates the window after the USE statement.

r-use.p

```
DEFINE VARIABLE w1 AS CHARACTER VIEW-AS TEXT FONT 0 FORMAT "x(34)"
  INITIAL "This is font 0 in the first window".
DEFINE VARIABLE w2 AS CHARACTER VIEW-AS TEXT FONT 0 FORMAT "x(35)"
  INITIAL "This is font 0 in the second window".
DEFINE VARIABLE new_win AS WIDGET-HANDLE.

LOAD "env1".
LOAD "env2".

USE "env1".

DISPLAY w1 WITH NO-LABELS WITH FRAME a.
PAUSE.

USE "env2".
CREATE WINDOW new_win.
CURRENT-WINDOW = new_win.
DISPLAY w2 in WINDOW new_win WITH NO-LABELS WITH FRAME b.
PAUSE.

DELETE WIDGET new_win.
```

This procedure depends on the existence of files named env1.ini and env2.ini, each of which contains a font definition for font0. If you run this procedure in your environment, you must create these files.

Notes

- Use this statement with applications (such as the User Interface Builder) that build and run other applications using a unique set of environment specifications.
- An application must use this statement after the LOAD statement and before a new window is created to make the loaded set of environment specifications apply to the new window.
- Subsequent PUT-KEY-VALUE and GET-KEY-VALUE statements apply to the environment made available by the USE statement.

See also GET-KEY-VALUE statement, LOAD statement, PUT-KEY-VALUE statement

USERID function

Returns the user ID of the current user.

Syntax

```
USERID [ ( logical-dbname ) ]
```

logical-dbname

The logical name of the database from which you want to retrieve the user ID. The logical database name must be a character string enclosed in quotes, or a character expression. If you do not specify this argument, the Compiler inserts the name of the database that is connected when the procedure is compiled. If you omit this argument and more than one database is connected, Progress returns a compiler error.

Example

This one-line procedure displays the current user ID for the database with the DICTDB alias:

r-userid.p

```
DISPLAY USERID("DICTDB") LABEL "You are logged in as"
  WITH SIDE-LABELS.
```

Notes

- Use the Userid (-U) parameter together with the Password (-P) parameter. Progress checks the _User table for the user ID supplied with the -U parameter. When it finds that user ID, it compares the password supplied with the -P parameter with the password in the _User table. If the two passwords match, Progress assigns that user ID to the OpenEdge session.
- When using the USERID function, Progress returns a compiler error under the following conditions:
  - There is no database connected.
  - You omit the *logical-dbname* argument and more than one database is currently connected.
- When specifying the *logical-dbname* argument, you must provide the name of the logical database, not the physical database.
- Every user who enters Progress is given an initial user ID.

  Table 51 shows how Progress determines a user's initial user ID on UNIX.

Table 51: Determining a UNIX user ID

Are there records in the _User table?	Are the -U and -P startup options supplied?	User ID
NO	YES	Error: -U and -P not allowed unless there are entries in the _User table.
NO	NO	UNIX user ID.
YES	NO	"" (blank user ID).
YES	YES	If the -U *userid* and -P *password* match those in the _User table, use that user ID. Otherwise, do not assign a user ID.

Table 52 shows how Progress determines a user's initial user ID in Windows.

Table 52: Determining a Windows user ID

Are there records in the _User table?	Are the -U and -P startup options supplied?	User ID
NO	YES	Error: -U and -P not allowed unless there are entries in the _User table.
NO	NO	"" (blank user ID) or operating system user ID if available.
YES	NO	"" (blank user ID.
YES	YES	If the -U *userid* and -P *password* match those in the _User table, use that user ID. Otherwise, do not assign a user ID.

- After Progress starts running, you can use the SETUSERID function to change the current user ID.
- Progress user IDs are case sensitive.
- See *OpenEdge Development: Programming Interfaces* and *OpenEdge Data Management: Database Administration* for more information on security.

See also CONNECT statement, SETUSERID function

VALID-EVENT function

Verifies whether a specified event is valid for a specified widget. For each type of widget, only certain events are valid. The function returns a value (TRUE/FALSE).

Note: Does not apply to SpeedScript programming.

Syntax

```
VALID-EVENT ( widget-handle , event-name [ , platform ] )
```

widget-handle

An expression that produces a value of type WIDGET-HANDLE. The value must be the handle of a valid widget.

event-name

A character-string expression that evaluates to the name of an event.

platform

A character-string expression that evaluates to the name of a platform type: GUI or TTY.

See also

LAST-EVENT system handle, LIST-EVENTS function, LIST-QUERY-ATTRS function, LIST-SET-ATTRS function, LIST-WIDGETS function

VALID-HANDLE function

Verifies that a handle is valid.

Note: Does not apply to SpeedScript programming.

Syntax

```
VALID-HANDLE ( handle )
```

handle

An expression that evaluates to a value of type HANDLE or WIDGET-HANDLE. If the handle represents an object that is currently valid, VALID-HANDLE returns TRUE. If the handle is no longer valid (if, for example, some procedure deleted the object), the function returns FALSE.

Example

In the following example, the user creates a window dynamically. The WINDOW-CLOSE trigger uses the VALID-HANDLE function to determine whether the window has been created.

r-valhnd.p

```
DEFINE VARIABLE mywin AS WIDGET-HANDLE.
DEFINE BUTTON mkwin   LABEL "New Window".

ENABLE mkwin.

ON CHOOSE OF mkwin
   DO:
      CREATE WINDOW mywin
          ASSIGN VISIBLE = TRUE
                 TITLE = "Second Window"
                 MAX-WIDTH-CHARS = 40
                 MAX-HEIGHT-CHARS = 10.
     SELF:SENSITIVE = FALSE.
   END.

ON WINDOW-CLOSE OF DEFAULT-WINDOW
   DO:
      IF VALID-HANDLE(mywin)
      THEN DELETE WIDGET mywin.
   END.

WAIT-FOR WINDOW-CLOSE OF DEFAULT-WINDOW.
```

In the example, the VALID-HANDLE function returns a TRUE value only if the window has been created (that is, mywin does not have the Unknown value (?)) and the window has not been deleted. Therefore, the DELETE WIDGET statement executes only if mywin is a valid widget handle.

Notes

- A widget handle becomes invalid if the associated widget or procedure is deleted or is out of scope.
- This function is useful when walking through a list of widgets or persistent procedures using the PREV-SIBLING or NEXT-SIBLING attributes.

  VALID-HANDLE(*handle*:PREV-SIBLING) is FALSE when you reach the first handle in the list. VALID-HANDLE(*handle*:NEXT-SIBLING) is FALSE when you reach the last handle in the list.
- The VALID-HANDLE function supports handles to AppServers, proxy persistent procedures, remote persistent procedures, and COM objects. For more information on AppServers, see *OpenEdge Application Server: Developing AppServer Applications*. For more information on COM objects, see *OpenEdge Development: Programming Interfaces*.

See also CREATE SAX-READER statement, WIDGET-HANDLE function

VALID-OBJECT function

Verifies that an object reference is for a valid class object instance. If the object reference represents a class object instance that is currently valid, the function returns TRUE. If the object reference is no longer valid (if, for example, some procedure or method deleted the object instance), the function returns FALSE.

Syntax

```
VALID-OBJECT ( object-reference )
```

object-reference

An object reference for a user-defined class object instance.

VALIDATE statement

Verifies that a record complies with mandatory field and unique index definitions.

Syntax

```
VALIDATE record [ NO-ERROR ]
```

record

The name of the record you want to validate.

To validate a record in a table defined for multiple databases, you must qualify the record's table name with the database name. See the Record phrase reference entry for more information.

NO-ERROR

Specifies that any errors that occur in the attempt to validate the record are suppressed. After the VALIDATE statement completes, you can check the ERROR-STATUS system handle for information on any errors that occurred.

Example

This procedure prompts for an item number. If an item with that number is not available, the procedure creates a new item record and lets you supply some item information. The VALIDATE statement checks the data you enter against the index and mandatory field criteria for the item record.

r-valid.p

```
REPEAT FOR item:
  PROMPT-FOR item-num.
  FIND item USING item-num NO-ERROR.
  IF NOT AVAILABLE item THEN DO:
    CREATE item.
    ASSIGN item-num.
    UPDATE item-name price.
    VALIDATE item.
  END.
  ELSE DISPLAY item-name price.
END.
```

Notes

- Because validation is done automatically, you rarely have to use the VALIDATE statement. Progress automatically validates a record when a record in the record buffer is replaced by another, a record's scope iterates or ends, the innermost iterating subtransaction block that creates a record iterates, or a transaction ends. For more information on record scoping and subtransaction blocks, see the chapter on block properties in *OpenEdge Development: Progress 4GL Handbook*.
- Progress automatically validates mandatory fields when those fields are modified.
- If the validation fails on a newly-created record, VALIDATE raises the ERROR condition.
- Progress performs validation when it leaves a field.
- For complex validations, it might be easier to use the IF...THEN...ELSE statement instead of the VALIDATE statement.
- You cannot use the VALIDATE statement to test fields that are referenced in SQL statements, since validation is not performed for these fields.
- If a field or table has been modified, the VALIDATE statement causes WRITE events and all related WRITE triggers to execute.

See also IF...THEN...ELSE statement

VIEW statement

Displays a widget (sets its VISIBLE attribute to TRUE).

Note: Does not apply to SpeedScript programming.

Syntax

```
VIEW
  [ STREAM stream ]
  [ widget-phrase ]
  [ IN WINDOW window ]
```

STREAM *stream*

Specifies the name of a stream. If you do not name a stream, Progress uses the unnamed stream.

widget-phrase

Specifies the widget you want to view. You can view windows, frames, and field-level widgets. You cannot view menus. If you do not use this option, VIEW sets the VISIBLE attribute for the default frame for the current block.

IN WINDOW *window*

Specifies the window in which to view the widget.

Example

The `r-view2.p` procedure displays information on a sales representative and then displays all the customers belonging to that sales representative. Each new sales representative is displayed on a new page. In addition, if the information for a sales representative takes up more than one page, a separate FORM statement describes a continuation header for that sales representative. The VIEW statement for the PAGE-TOP frame hdr2, activates the header for subsequent page breaks.

r-view2.p

```
OUTPUT TO slsrep PAGED PAGE-SIZE 10.

FOR EACH salesrep:
  PAGE.
  FORM HEADER "Sales rep report" "Page" AT 60 PAGE-NUMBER FORMAT ">>>9".

  DISPLAY SKIP(1) sales-rep rep-name region WITH NO-LABELS.

  FORM HEADER "Sales rep report" sales-rep "(continued)"
    "Page" AT 60 PAGE-NUMBER FORMAT ">>>9" SKIP(1)
    WITH FRAME hdr2 PAGE-TOP.

  VIEW FRAME hdr2.

  FOR EACH customer OF salesrep:
    DISPLAY cust-num name address city state.
  END.
END.
```

Notes

- If the widget is already visible, the VIEW statement has no effect.
- Viewing a widget does not, by itself, show any of its data. To view data in a widget, you must use a data display statement (such as DISPLAY) or assign the data directly to the widget's SCREEN-VALUE attribute.
- When you view a window, its frames and their descendant widgets are not displayed, unless you explicitly view or display them.
- When you view a widget, Progress displays that widget unless its parent window or an ancestor window has its HIDDEN attribute set to TRUE.
- When you view a widget that has its HIDDEN attribute set to TRUE, Progress sets the widget's HIDDEN attribute to FALSE.
- When you view a widget contained by a window that is invisible (VISIBLE attribute is FALSE), that widget and the containing window is displayed unless the containing window's HIDDEN attribute is set to TRUE.

- When you view a widget contained by one or more ancestor frames that are invisible, the VISIBLE attribute is set to TRUE and the HIDDEN attribute is set to FALSE for both the viewed widget and all its ancestor frames. However, if the containing window or an ancestor window has its HIDDEN attribute set to TRUE, neither the viewed widget nor its ancestor frames are displayed.
- When you view a frame, that frame and all widgets contained within it are displayed except those widgets whose HIDDEN attributes are set to TRUE.
- When you view a window, Progress displays that window and any ancestor windows only if no ancestor window has its HIDDEN attribute set to TRUE. If Progress displays the window, it also views any descendant windows down to, but not including, the first descendent window that has its HIDDEN attribute set to TRUE.
- If you are displaying a root frame and there is not enough room in the window for the new root frame to display, Progress removes other root frames, starting from the bottom of the window, until there is room for the new root frame.
- In the case of a PAGE-TOP or PAGE-BOTTOM frame, the VIEW statement activates the frame for display at the beginning or end of each page.

See also HIDE statement, Widget phrase

VIEW-AS phrase

Defines a static widget to represent a field or variable on the screen.

Note: Does not apply to SpeedScript programming.

Syntax

```
VIEW-AS
  {    combo-box-phrase
     | editor-phrase
     | FILL-IN
         [ NATIVE ]
         [ size-phrase ]
         [ TOOLTIP tooltip ]
     | radio-set-phrase
     | selection-list-phrase
     | slider-phrase
     | TEXT
         [ size-phrase ]
         [ TOOLTIP tooltip ]
     | TOGGLE-BOX
         [ size-phrase ]
         [ TOOLTIP tooltip ]
  }
```

combo-box-phrase

Specifies that a field or variable is viewed as a combo box widget. You can use a combo box to represent a value of any data type. This is the syntax for *combo-box-phrase*:

```
VIEW-AS COMBO-BOX
  [ LIST-ITEMS item-list | LIST-ITEM-PAIRS item-pair-list ]
  [ INNER-LINES lines ] [ size-phrase ] [ SORT ]
  [ TOOLTIP tooltip ]
  [ SIMPLE | DROP-DOWN | DROP-DOWN-LIST ]
  [ MAX-CHARS characters ]
  [ AUTO-COMPLETION [ UNIQUE-MATCH ] ]
```

For more information, see the COMBO-BOX phrase reference entry.

editor-phrase

Specifies that a CHARACTER or LONGCHAR field or variable is viewed as a text editor widget. A text editor widget supports cut, paste, word-wrap, and auto-indent features. This is the syntax for *editor-phrase*:

```
EDITOR
  {     size-phrase
      | INNER-CHARS char INNER-LINES lines
  }
  [ BUFFER-CHARS chars ]
  [ BUFFER-LINES lines ]
  [ LARGE ]
  [ MAX-CHARS characters ]
  [ NO-BOX ]
  [ NO-WORD-WRAP ]
  [ SCROLLBAR-HORIZONTAL ]
  [ SCROLLBAR-VERTICAL ]
  [ TOOLTIP tooltip ]
```

For more information, see the EDITOR phrase reference entry.

```
FILL-IN [ NATIVE ] [ size-phrase ]
```

Specifies that the field or variable is viewed as a fill-in widget. In a fill-in field, the literal value of the field or variable is displayed. On update, the user types the literal value into the fill-in field.

You can specify FILL-IN for any CHARACTER, INTEGER, DECIMAL, DATE, DATETIME, DATETIME-TZ, or LOGICAL value (with or without extents). FILL-IN is the default representation for those values.

Note that Windows allows a user to transfer focus to the fill-in field by pressing **ALT** and one of the letters in the label. For more information on specifying a label using the LABEL option, see the Format phrase reference entry.

If you specify NATIVE, then the field behaves like a native fill-in field under the current user interface. A non-NATIVE field behaves like a default Progress 4GL fill-in field under any interface. Native fill-in fields provide better consistency with other applications in graphical environments, but do not support some Progress constructs such as the UPDATE statement with the TEXT option or the CHOOSE statement.

When a non-NATIVE (Progress 4GL) fill-in is disabled, the border disappears, but the text does not gray out. When a NATIVE fill-in is disabled, the text grays out.

Like the other static widgets that can be defined using the VIEW-AS phrase, you can specify ToolTips for the fill-in widget using the TOOLTIP option.

radio-set-phrase

Specifies that the field or variable is viewed as a radio set widget. A radio button set is a series of buttons, of which only one can be TRUE at a time. When the user sets one of the buttons to TRUE, the others are set to FALSE. You can specify a *radio-set-phrase* for any group of CHARACTER, INTEGER, DECIMAL, DATE, or LOGICAL values (with or without extents). This is the syntax for *radio-set-phrase*:

```
RADIO-SET
  [ HORIZONTAL [ EXPAND ] | VERTICAL ]
  [ size-phrase ]
  RADIO-BUTTONS label , value [ , label, value ... ]
  [ TOOLTIP tooltip ]
```

Note: If two or more buttons of a radio set use the same label, the Progress 4GL uses only the value of the first button.

For more information, see the RADIO-SET phrase reference entry.

selection-list-phrase

Specifies that the field or variable is viewed as a selection list widget. You can only specify the *selection-list-phrase* for a character-string value. A selection list is a scrollable list of CHARACTER values. If the field is enabled for input, the user can select one or more values from the list.

```
SELECTION-LIST
  [ SINGLE | MULTIPLE ]
  [ NO-DRAG ]
  [ LIST-ITEMS item-list ]
  [ SCROLLBAR-HORIZONTAL ]
  [ SCROLLBAR-VERTICAL ]
  { size-phrase | INNER-CHARS cols INNER-LINES rows }
  [ SORT ]
  [ TOOLTIP tooltip ]
```

For more information, see the SELECTION-LIST phrase reference entry.

slider-phrase

Specifies that the field or variable is viewed as a slider. Specify the *slider-phrase* for an integer value only. A slider is a graphical representation of a numeric range. It is composed of a rectangular area that contains a trackbar. You can change the current value within a defined range by moving the pointer that resides on the trackbar.

```
VIEW-AS SLIDER
  MAX-VALUE max-value MIN-VALUE min-value
  [ HORIZONTAL | VERTICAL ]
  [ NO-CURRENT-VALUE ]
  [ LARGE-TO-SMALL ]
  [ TIC-MARKS
      { NONE | TOP | BOTTOM | LEFT | RIGHT | BOTH }
      [ FREQUENCY n ]
  ]
  [ TOOLTIP tooltip ]
  [ size-phrase ]
```

For more information, see the SLIDER phrase reference entry.

TEXT [*size-phrase*]

Specifies that the field or variable is viewed as read-only text. In a graphical environment, a text field takes up less space on the screen than a native fill-in field.

You can specify TEXT for any CHARACTER, INTEGER, DECIMAL, DATE, or LOGICAL value (with or without extents).

TOGGLE-BOX [*size-phrase*]

Specifies that the field or variable is viewed as a toggle box widget. A toggle box is a small box that is either marked or not marked to indicate a TRUE or FALSE value, respectively. You can specify TOGGLE-BOX for any LOGICAL value.

Note that Windows allows a user to select a toggle-box item by pressing **ALT** and one of the letters in the side label. For more information on specifying a label using the LABEL option, see the Format phrase reference entry.

`TOOLTIP` *`tooltip`*

Allows you to define a help text message for a toggle box. Progress automatically displays this text when the user pauses the mouse over the toggle-box.

You can add or change the TOOLTIP option at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed. No ToolTip is the default. The TOOLTIP option is supported in Windows only.

Example

The following procedure defines a character variable and views it in succession as a text widget, a fill-in widget, an editor widget, and finally as a text widget again. The procedure shows that you can represent a character variable in several ways, as long as each representation appears in a separate frame.

r-viewas.p

```
DEFINE VARIABLE test   AS CHARACTER INITIAL "Now is the time"
          FORMAT "x(30)".

DISPLAY test VIEW-AS TEXT LABEL "Labels cannot be changed"
  WITH FRAME a SIDE-LABELS.

PAUSE.

UPDATE test VIEW-AS FILL-IN
      LABEL "But fillins can, please enter a new value"
       WITH FRAME b SIDE-LABELS.

UPDATE test VIEW-AS EDITOR
  INNER-CHARS 16 INNER-LINES 2 MAX-CHARS 70
  LABEL  "As can editors, please enter a new value:"
  WITH FRAME c.

DISPLAY test VIEW-AS TEXT FORMAT "x(70)"
  LABEL "The final value is:"
  WITH FRAME d.
```

For additional examples, see the COMBO-BOX phrase, EDITOR phrase, RADIO-SET phrase, SELECTION-LIST phrase, and SLIDER phrase reference entries.

Notes

- To create a static widget, you must define a static frame that contains the widget. Each frame you define that contains the widget creates an additional instance of that widget for the underlying field or variable. The widget handle for a static widget is not available until the widget is created.
- You can also use the VIEW-AS option in the Frame phrase and MESSAGE statement to indicate a dialog box and alert box, respectively.
- In Windows, if no font is specified for a fill-in field, Progress uses two default fonts:
  - A fixed font for date fields, numeric fields, and character fields that contain fill characters (such as the parentheses surrounding the area code of a telephone number).
  - A proportional font for character fields that do not contain fill characters.

  Progress looks for these fonts in the current environment, which may be the registry (Windows only) or an initialization file. If the current environment does not define these fonts, Progress uses the system default fixed and proportional fonts. For more information on environments, see *OpenEdge Deployment: Managing 4GL Applications*.

See also COMBO-BOX phrase, EDITOR phrase, RADIO-SET phrase, SELECTION-LIST phrase, SIZE phrase, SLIDER phrase

WAIT-FOR statement

The WAIT-FOR statement instructs Progress to stop executing the current block until a specific Progress event occurs. Progress continues to respond to all other incoming events and execute any associated triggers or event procedures while in this wait state.

Syntax

```
WAIT-FOR event-list OF widget-list
  [ OR event-list OF widget-list ] ...
  [ FOCUS widget ]
  [ PAUSE n ]
```

```
WAIT-FOR "WEB-NOTIFY"
OF DEFAULT-WINDOW
  [ PAUSE n ]
  [ EXCLUSIVE-WEB-USER ]
```

event-list

A space or comma-separated list of user-interface events and other Progress events to wait for.

An event can be any event described in the "Events Reference" section on page 2171.

widget-list

A space- or comma-separated list of widgets with which the event is associated. For more information on referencing widgets, see the Widget phrase reference entry.

FOCUS *widget*

Specifies the widget that initially receives input focus when the WAIT-FOR statement is executed. The value *widget* must be a valid reference to a widget (a widget name or handle) that is currently displayed and enabled.

PAUSE *n*

Specifies a time-out interval for the WAIT-FOR statement. The value *n* can be any numeric expression. If a period of *n* seconds elapses between events, the WAIT-FOR automatically terminates.

Examples

This procedure defines two buttons, defines triggers for them, and enables them. The procedure then waits for the user to close the current window. The initial focus is placed on the button labeled MORE. The user can then choose buttons continuously until closing the window or exiting with the **END-ERROR** key.

r-wait.p

```
DEFINE BUTTON more-button LABEL "MORE".
DEFINE BUTTON next-button LABEL "NEXT".

FORM customer.cust-num customer.name more-button next-button
    WITH FRAME brief.

FORM customer EXCEPT cust-num name
    WITH FRAME full.

ON CHOOSE OF more-button
   DISPLAY customer EXCEPT cust-num name WITH FRAME full.

ON CHOOSE OF next-button
   DO:
      HIDE FRAME full.
      FIND NEXT customer NO-ERROR.
      IF AVAILABLE customer
      THEN DISPLAY customer.cust-num customer.name WITH FRAME brief.
   END.

FIND FIRST customer.
DISPLAY customer.cust-num customer.name WITH FRAME brief.

ENABLE more-button next-button WITH FRAME brief.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW FOCUS more-button.
```

If the user closes the current window then execution continues after the WAIT-FOR statement. In this case, the procedure ends because there are no more statements.

The following procedure uses the PAUSE option of the WAIT-FOR statement so that you automatically jump ahead to the next record if the user does not perform any action within three seconds after the customer information is displayed:

r-waitpn.p

```
DEFINE BUTTON more-button LABEL "MORE".
DEFINE BUTTON next-button LABEL "NEXT".
DEFINE VARIABLE jump-ahead AS LOGICAL INITIAL TRUE.

FORM customer.cust-num customer.name more-button next-button
    WITH FRAME brief.FORM customer EXCEPT cust-num name
    WITH FRAME full.

ON CHOOSE OF more-button
   DO:
      DISPLAY customer EXCEPT cust-num name WITH FRAME full.
      jump-ahead = FALSE.
   END.

ON CHOOSE OF next-button
   DO:
     jump-ahead = TRUE.
   END.

ON WINDOW-CLOSE OF CURRENT-WINDOW
   DO:
     QUIT.
   END.

ENABLE more-button next-button WITH FRAME brief.

DO WHILE TRUE:
   IF jump-ahead
   THEN RUN next-cust.

   WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW OR CHOOSE OF next-button
            FOCUS more-button PAUSE 3.
END.

PROCEDURE next-cust:
   HIDE FRAME full.
   FIND NEXT customer NO-ERROR.
   IF AVAILABLE customer
   THEN DISPLAY customer.cust-num customer.name WITH FRAME brief.
END.
```

In this example, the code for find the next customer has been moved to an internal procedure. The WAIT-FOR statement has been placed inside a DO loop. The loop iterates when the user chooses the NEXT button or three seconds elapse. (If the user closes the window, the QUIT statement is executed and the loop does not iterate.) On each iteration, if the variable jump-ahead is TRUE, then the next-cust procedure is run to find and display the next customer. If the user chooses the MORE button for a customer, jump-ahead is set to FALSE. This prevents the procedure from automatically jumping ahead to the next customer. Instead, the user can spend time examining the data. To move ahead to the next customer, the user must explicitly choose the NEXT button. At that point, jump-ahead is reset to TRUE.

Notes

- Any widget associated with an event must be enabled before you wait on it.
- In general, do not use an UPDATE statement in an application that executes a WAIT-FOR statement. One exception is updating fields in a dialog box. For more information, see the chapter on programming models in *OpenEdge Development: Progress 4GL Handbook*.
- In general, if you nest two WAIT-FOR statements in a single OpenEdge application (where the nested WAIT-FOR executes in a trigger), you must ensure that your application satisfies the nested WAIT-FOR first. The event that satisfies the outer WAIT-FOR statement should be the terminating event for your application.
- In general, when a modal dialog box is active, the *event-list* can reference only events supported by the active dialog box and the widgets it contains. There are two exceptions:
  - You can specify an event on a procedure handle as long as *widget-list* specifies only a single procedure handle.
  - You can specify the PROCEDURE-COMPLETE event on an asynchronous request handle.
- The PROCEDURE-COMPLETE event occurs for an asynchronous request handle when the current OpenEdge session receives the response message from the AppServer that executed the request. When the WAIT-FOR statement executes, it processes any PROCEDURE-COMPLETE event that has occurred but has not yet been processed.
- PROCEDURE-COMPLETE events from a single AppServer connection are processed in the order that the associated asynchronous requests were originally generated. To ensure that all pending PROCEDURE-COMPLETE events are handled by a single WAIT-FOR statement, specify a single PROCEDURE-COMPLETE event for the last asynchronous request handle generated before the WAIT-FOR statement.
- To process a PROCEDURE-COMPLETE event for a particular asynchronous request handle, Progress:

- Decrements the ASYNC-REQUEST-COUNT attribute for the server referenced by SERVER attribute for the asynchronous request handle.
- Decrements the ASYNC-REQUEST-COUNT attribute for a persistent procedure, if the PERSISTENT-PROCEDURE attribute of the asynchronous request handle refers to a valid persistent procedure.
- Sets the COMPLETE attribute for the asynchronous request handle to TRUE.
- Sets the STOP, QUIT, and ERROR attributes for the asynchronous request handle appropriately as indicated by the response message from the AppServer.
- Sets the return value for the RETURN-VALUE function, if a return value was returned by the AppServer.
- Stores any error information returned from the AppServer in the ERROR-STATUS system handle.
- Attempts to execute the event procedure specified by the EVENT-PROCEDURE and the EVENT-PROCEDURE-CONTEXT attributes for the asynchronous request handle, if EVENT-PROCEDURE is not the empty string (“”).
- Sets each INPUT parameter for the event procedure to the Unknown value (?) or, if the parameter is a TEMP-TABLE, the TEMP-TABLE remains unchanged, if the response message indicates that the remote request finished with a STOP, ERROR, or QUIT condition.
- Sets the INPUT parameter values for the event procedure to the OUTPUT and INPUT-OUTPUT parameter values returned by the remote procedure, if the response message indicates that the remote request completed successfully.
- Displays an error message, if a specified event procedure fails to execute for any reason.
- Raises any unhandled STOP condition, ERROR condition, or QUIT condition in the context of the WAIT-FOR statement, if the event procedure completes execution with that condition.

- These are possible causes for failing to execute the event procedure for a PROCEDURE-COMPLETE event. All of these failures raise a STOP condition in the context of the WAIT-FOR statement:
  - The procedure handle referenced by the EVENT-PROCEDURE-CONTEXT attribute is no longer valid.
  - The internal procedure specified by the EVENT-PROCEDURE attribute cannot be found.
  - The parameters to the internal procedure specified by the EVENT-PROCEDURE attribute are not all INPUT parameters.
  - The parameter signature of the internal procedure specified by the EVENT-PROCEDURE attribute does not match the output parameters returned in the response message for the asynchronous request.
- For SpeedScript, the WAIT-FOR statement instructs WebSpeed to stop executing the current block until the WEB-NOTIFY event occurs. The WEB-NOTIFY event is intended for internal use only, it does not apply to SpeedScript programming.

See also DISABLE statement, ENABLE statement, ON statement, Trigger phrase, Widget phrase

WEEKDAY function

Evaluates a date expression and returns the day of the week as an integer from 1 (Sunday) to 7 (Saturday) for that date.

Syntax

```
WEEKDAY ( date )
```

```
WEEKDAY ( datetime-expression )
```

date

A date expression for which you want the day of the week.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The WEEKDAY function returns the weekday of the date part of the DATETIME or DATETIME-TZ value.

Example This procedure tells you the day of the week that you were born and how many days old you are:

r-wkday.p

```
DEFINE VARIABLE birth-date AS DATE
  LABEL "Birth Date".
DEFINE VARIABLE daynum AS INTEGER.
DEFINE VARIABLE daylist AS CHARACTER FORMAT "x(9)"
  INITIAL "Sunday, Monday, Tuesday, Wednesday, Thursday,
      Friday, Saturday".
DEFINE VARIABLE dayname AS CHARACTER
  LABEL "Day You Were Born".
DEFINE VARIABLE daysold AS INTEGER
  LABEL "Days Since You Were Born".

REPEAT:
  SET birth-date.
  daynum = WEEKDAY(birth-date).
  dayname = ENTRY(daynum,daylist).
  daysold = TODAY - birth-date.
  DISPLAY dayname daysold.
END.
```

See also DAY function, MONTH function, NOW function, TODAY function

WIDGET-HANDLE function

Converts a string representation of a widget handle to a valid widget handle.

Syntax

```
WIDGET-HANDLE ( widget-handle-string )
```

Caution: Use this function only to convert a widget handle previously stored as a string value back to a valid widget handle. If you convert an arbitrary string to widget handle using this function and then reference the new widget handle, a system error will occur. The VALID-HANDLE function references a widget handle to validate the handle. If you use the VALID-HANDLE function to validate a widget handle generated from an arbitrary string value, a system error will occur.

widget-handle-string

A string representation of a widget handle. Since widget handles are integer values, the string must contain only numeric characters.

Example

The following procedure creates a frame, stores the widget handle of the frame as a string value, deletes the frame, converts the string representation of the frame widget handle back to a widget handle, and then tests if the widget handle is valid:

r-widhd.p

```
DEFINE VARIABLE whand AS WIDGET-HANDLE.
DEFINE VARIABLE chand AS CHARACTER.

CREATE FRAME whand.
chand = STRING(whand).
DELETE WIDGET whand.
whand = WIDGET-HANDLE(chand).
MESSAGE VALID-HANDLE(whand) VIEW-AS ALERT-BOX
  INFORMATION BUTTONS OK.
```

The VALID-HANDLE function returns a FALSE value because the frame was deleted and the widget handle is no longer valid.

Notes

- The WIDGET-HANDLE function can convert the string representation of procedure and system handles, as well as widget handles.
- For SpeedScript, the only valid use is to convert the handle of a QUERY object that you create using the CREATE WIDGET statement.

See also

CREATE widget statement, DATE function, DECIMAL function, INTEGER function, STRING function, VALID-HANDLE function

Widget phrase

References a widget in a statement. The Widget phrase is used in the APPLY, ON, and WAIT-FOR statements.

Note: Does not apply to SpeedScript programming.

Syntax

```
{     FRAME frame
    | [ FIELD ] field [ IN FRAME frame ]
    | column [ IN BROWSE browse ]
    | { MENU | SUB-MENU } menu
    | MENU-ITEM menu-item [ IN MENU menu ]
    | handle
    | system-handle
}
```

`FRAME frame`

Specifies a frame widget. The *frame* parameter must be the name of an existing frame.

`[ FIELD ] field [ IN FRAME frame ]`

Specifies a field. The FIELD keyword is optional. The *field* parameter must be the name of an existing field-level widget: a fill-in, editor, text, slider, toggle box, radio set, selection list, combo box, button, image, rectangle, or browse. Use the IN FRAME option to qualify the widget, if necessary.

`column [ IN BROWSE browse ]`

Specifies a column or cell in a browse widget. Use the IN BROWSE option to qualify the widget, if necessary. For more information on when you can reference browse columns and cells, see the DEFINE BROWSE statement reference entry.

`{ MENU | SUB-MENU } menu`

Specifies a menu or submenu. The *menu* parameter must be the name of an existing menu. The menu can be a pop-up menu, pull-down menu, or menu bar. Within the widget phrase, Progress does not distinguish between MENU and SUB-MENU.

`MENU-ITEM` *menu-item* [`IN MENU` *menu*]

Specifies an menu item within a menu. The menu item parameter must be the name of an existing menu item. Use the IN MENU option to qualify the menu item, if necessary.

handle

Variable or field that specifies a valid widget, procedure, or system handle.

system-handle

Specifies a built-in system handle. The system handle parameter must be one of the built-in system handles, which Table 53 lists.

Table 53: System handles *(1 of 2)*

System handle	Description
ACTIVE-WINDOW	A handle to the Progress window that has most recently received input focus during the session.
CLIPBOARD	A handle to the system clipboard.
COLOR-TABLE	A handle to information on the current color table.
COMPILER	A handle to information on the most recently executed COMPILE statement.
CURRENT-WINDOW	A setable handle to the default window for the OpenEdge session.[1,2]
DEBUGGER	A handle to the Application Debugger.
DEFAULT-WINDOW	A handle to the static window created by Progress for the session. Every session has one static window.[1]
ERROR-STATUS	A handle to information on the last statement executed with the NO-ERROR option.
FILE-INFO	A handle to information on an operating system file.
FOCUS	A handle to the field-level widget that currently has keyboard focus (that is, the current field).
FONT-TABLE	A handle to information on the current font table.

Table 53: System handles *(2 of 2)*

System handle	Description
LAST-EVENT	A handle to the last event received by the program.
RCODE-INFO	A handle to information on a Progress r-code file.
SELF	A handle for the widget associated with the currently executing user-interface trigger.
SESSION	A handle to information on the current OpenEdge session.
SOURCE-PROCEDURE	A handle to the procedure file that contains the original invocation (RUN statement or function invocation) of the current internal procedure or user-defined function.
TARGET-PROCEDURE	From within an internal procedure: A handle to the procedure file mentioned, explicitly or implicitly, by the original RUN statement that invoked (perhaps through a chain of super procedures) the current internal procedure. From within a user-defined function: A handle to the procedure file mentioned, explicitly or implicitly, by the original function invocation that invoked (perhaps through a chain of super versions of functions) the current user-defined function.
THIS-PROCEDURE	A handle to the executing external procedure in which the handle is referenced.

[1] The initial setting of the CURRENT-WINDOW handle is the Progress static window. CURRENT-WINDOW can also be set to the handle of any dynamic window.

[2] If the THIS-PROCEDURE:CURRENT-WINDOW attribute is set to the handle of a valid window, this window becomes the default window for the executing procedure (overriding the setting of the CURRENT-WINDOW handle). The setting of THIS-PROCEDURE:CURRENT-WINDOW changes the default window only for the current external procedure block.

Note For information on how to specify widgets for attribute and method references, see the chapter on widgets and handles in *OpenEdge Development: Progress 4GL Handbook*.

YEAR function

Evaluates a date expression and returns the year value of that date, including the century.

Syntax

```
YEAR ( date )
```

```
YEAR ( datetime-expression )
```

date

A date expression for which you want to determine the year.

datetime-expression

An expression that evaluates to a DATETIME or DATETIME-TZ. The YEAR function returns the year of the date part of the DATETIME or DATETIME-TZ value.

Example

This procedure uses the YEAR function to determine if an order date is in this century or the next, and then uses a different display format for each:

r-year.p

```
DEFINE VARIABLE outfmt AS CHARACTER.
DEFINE VARIABLE orddate AS CHARACTER
  LABEL "Order Date" FORMAT "x(10)".

FOR EACH order:
  IF YEAR(odate) >= 2000
  THEN outfmt = "99/99/9999".
  ELSE outfmt = "99/99/99".
  orddate = STRING(odate,outfmt).
  DISPLAY order.order-num orddate terms.
END.
```

See also

YEAR-OFFSET attribute

Widget Reference

This chapter contains reference entries for the Progress user-interface widgets. For more information on the attributes, methods, and events listed for each widget, see the appropriate sections in this manual.

You may consider a user-interface widget to be supported for all interfaces and on all operating systems unless otherwise indicated in the reference entry. These user-interface widgets do not apply to SpeedScript programming.

Because widgets are not realized in batch mode, you cannot use any method or attribute that requires the widget to be realized in batch mode.

Note: Of the common attributes listed for the following widgets, BGCOLOR, FGCOLOR, FONT, MOVABLE, RESIZABLE, and SELECTABLE apply only to graphical interfaces; DCOLOR and PFCOLOR apply only to character interfaces. In character interfaces, all attributes and methods that reference pixels (for example HEIGHT-PIXELS) use a system default pixel value for the equivalent value in characters.

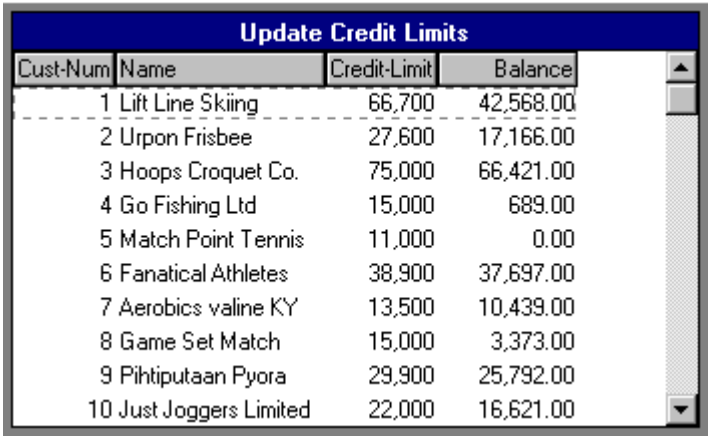
BROWSE widget

A browse widget lets you see data and select records from all the records associated with a database query. You can define a static browse widget with the DEFINE BROWSE statement or a dynamic browse widget with the CREATE BROWSE statement. The CREATE BROWSE is valid only in a graphical interface. A browse can be either a read-only tool for browsing through records, or it can be an editing tool for updating records, depending on the options you specify.

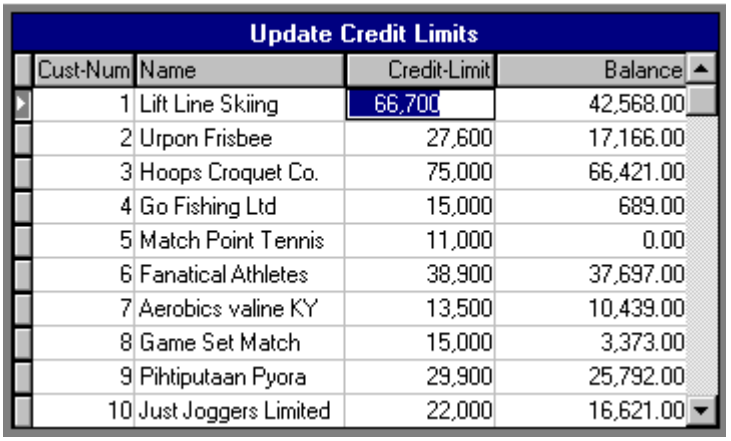
You can move and resize the browse and its components. Specifically, in graphical interfaces, you can move and resize the browse, move and change the width of the browse-column, and change the height of the browse-row. You can do all this through direct manipulation (by pointing, clicking, and dragging) and through the 4GL. For more information, see *OpenEdge Development: Progress 4GL Handbook*.

You can also use the mouse wheel to scroll the browse widget horizontally and vertically in Windows. When you rotate the mouse wheel up and down, the browse scrolls vertically up and down. When you rotate the mouse wheel up and down while holding down the **CTRL** key, the browse scrolls horizontally left and right. You can specify the number of rows the browse scrolls up and down per click of the mouse wheel on the Wheel tab in the Windows Mouse Properties dialog box (accessed through the Windows Control Panel).

The following figure shows a read-only browse widget:



The following figure shows an updateable browse. Note the inline editing capability in the focused row:



Update Credit Limits

Cust-Num	Name	Credit-Limit	Balance
1	Lift Line Skiing	66,700	42,568.00
2	Urpon Frisbee	27,600	17,166.00
3	Hoops Croquet Co.	75,000	66,421.00
4	Go Fishing Ltd	15,000	689.00
5	Match Point Tennis	11,000	0.00
6	Fanatical Athletes	38,900	37,697.00
7	Aerobics valine KY	13,500	10,439.00
8	Game Set Match	15,000	3,373.00
9	Pihtiputaan Pyora	29,900	25,792.00
10	Just Joggers Limited	22,000	16,621.00

Attributes

When describing browse attributes, it is important to understand the scope of each attribute. An attribute can apply to:

- The browse widget as a whole.
- A single browse column.
- A single browse cell. In this case, the attribute applies to only the single cell at the intersection of the named column and the focused row.
- Both the browse as a whole and a cell or column. For example, in the same trigger, you could change the background color of the whole browse to blue and the background color of the current cell to yellow.

When you want to reference an attribute that applies to the browse as a whole, the correct syntax is as follows:

```
browse-name:attribute-name IN FRAME frame-name  (for a static browse)
OR
browse-handle:attribute-name      (for a dynamic or static browse)
```

The IN FRAME qualifier is only necessary for a static browse to avoid ambiguity.

When an attribute applies to a column or a cell, the identifier is the field or variable name as listed in the DEFINE BROWSE statement. This identifier is known as the column name. The browse column's widget-handle may also be used. Here is the syntax:

```
column-name:attribute-name IN BROWSE browse-name  (static browse column)
OR
column-handle:attribute-name       (dynamic or static browse column)
```

The IN BROWSE qualifier is only necessary for a static browse to avoid ambiguity, but it is good programming practice to always include it, especially when you reference the same field as a separate widget type.

The following table lists all the attributes for the browse widget, whether they are readable and writeable, and their scope:

(1 of 6)

Attribute	Applies to
ALLOW-COLUMN-SEARCHING attribute[3]	Browse
AUTO-RESIZE attribute[3]	Column
AUTO-VALIDATE attribute	Column
AUTO-ZAP attribute	Cell
BGCOLOR attribute[1]	Browse, cell
BUFFER-FIELD attribute	Column
COLUMN attribute	Browse, cell
COLUMN-BGCOLOR attribute	Column
COLUMN-DCOLOR attribute[2]	Column

(2 of 6)

Attribute	Applies to
COLUMN-FGCOLOR attribute	Column
COLUMN-FONT attribute	Column
COLUMN-MOVABLE attribute[3]	Browse
COLUMN-PFCOLOR attribute[2]	Column
COLUMN-READ-ONLY attribute	Column
COLUMN-RESIZABLE attribute[3]	Browse
COLUMN-SCROLLING attribute	Browse
CONTEXT-HELP-ID attribute	Browse
CURRENT-COLUMN attribute	Browse
CURRENT-ROW-MODIFIED attribute	Browse
CURSOR-OFFSET attribute	Cell
DATA-TYPE attribute	Column
DCOLOR attribute[2]	Browse, cell
DISABLE-AUTO-ZAP attribute	Column
DOWN attribute[3]	Browse
DROP-TARGET attribute	Browse
DYNAMIC attribute	Browse
EDIT-CAN-PASTE attribute[1,3]	Column
EDIT-CAN-UNDO attribute	Column
EXPANDABLE attribute[3]	Browse
FGCOLOR attribute[1]	Browse, cell
FIRST-COLUMN attribute	Browse
FIT-LAST-COLUMN attribute	Browse
FOCUSED-ROW attribute	Browse
FOCUSED-ROW-SELECTED attribute	Browse
FONT attribute[1]	Browse, cell

(3 of 6)

Attribute	Applies to
FRAME attribute	Browse
FRAME-COL attribute	Browse
FRAME-NAME attribute	Browse
FRAME-ROW attribute	Browse
FRAME-X attribute	Browse
FRAME-Y attribute	Browse
HANDLE attribute	Browse, cell
HEIGHT-CHARS attribute[3]	Browse, cell
HEIGHT-PIXELS attribute[3]	Browse, cell
HELP attribute	Browse, column
HIDDEN attribute	Browse
HWND attribute[3]	Browse
INPUT-VALUE attribute	Cell
LABEL attribute	Column
LABELS attribute[3]	Browse
LABEL-BGCOLOR attribute	Column
LABEL-DCOLOR attribute[2]	Column
LABEL-FGCOLOR attribute	Column
LABEL-FONT attribute	Column
MAX-DATA-GUESS attribute	Browse
MENU-KEY attribute	Browse
MENU-MOUSE attribute[1]	Browse
MIN-COLUMN-WIDTH-CHARS attribute	Browse
MIN-COLUMN-WIDTH-PIXELS attribute	Browse
MODIFIED attribute	Browse, column
MOUSE-POINTER attribute	Browse, column

(4 of 6)

Attribute	Applies to
MOVABLE attribute[1,3]	Browse, column
MULTIPLE attribute	Browse
NAME attribute	Browse, cell
NEW-ROW attribute	Browse
NEXT-COLUMN attribute	Column
NEXT-SIBLING attribute	Browse
NEXT-TAB-ITEM attribute	Browse
NO-EMPTY-SPACE attribute	Browse
NO-VALIDATE attribute	Browse
NUM-COLUMNS attribute	Browse
NUM-DROPPED-FILES attribute	Browse
NUM-ITERATIONS attribute	Browse
NUM-LOCKED-COLUMNS attribute	Browse
NUM-SELECTED-ROWS attribute	Browse
NUM-VISIBLE-COLUMNS attribute	Browse
PARENT attribute	Browse
PFCOLOR attribute[2]	Cell
POPUP-MENU attribute	Browse
PREV-COLUMN attribute	Column
PREV-SIBLING attribute	Browse
PREV-TAB-ITEM attribute	Browse
PRIVATE-DATA attribute	Browse, column
QUERY attribute	Browse
READ-ONLY attribute	Browse, column
REFRESHABLE attribute	Browse
RESIZABLE attribute[1,3]	Browse, column
ROW attribute	Browse, cell

(5 of 6)

Attribute	Applies to
ROW-HEIGHT-CHARS attribute[3]	Browse
ROW-HEIGHT-PIXELS attribute[3]	Browse
ROW-RESIZABLE attribute[3]	Browse
ROW-MARKERS attribute	Browse
SCREEN-VALUE attribute	Cell
SCROLLBAR-VERTICAL attribute[3]	Browse
SELECTABLE attribute[1,3]	Browse
SELECTED attribute[1,3]	Browse
SELECTION-END attribute	Column
SELECTION-START attribute	Column
SELECTION-TEXT attribute	Column
SENSITIVE attribute	Browse
SEPARATORS attribute[3]	Browse
SEPARATOR-FGCOLOR attribute[3]	Browse
TAB-POSITION attribute	Browse
TAB-STOP attribute	Browse
TABLE attribute	Column
TEXT-SELECTED attribute	Column
TITLE attribute	Browse
TITLE-BGCOLOR attribute[1]	Browse
TITLE-DCOLOR attribute[2]	Browse
TITLE-FGCOLOR attribute[1]	Browse
TITLE-FONT attribute[1]	Browse
TOOLTIP attribute[1,3]	Browse
TYPE attribute	Browse, cell
VIEW-FIRST-COLUMN-ON-REOPEN attribute	Browse

(6 of 6)

Attribute	Applies to
VISIBLE attribute[4]	Browse, column
WIDGET-ID attribute [1,3]	Browse
WIDTH-CHARS attribute[3]	Browse, column
WIDTH-PIXELS attribute[3]	Browse, column
WINDOW attribute	Browse
X attribute	Browse, cell
Y attribute	Browse, cell

1 Graphical interfaces only.

2 Character interfaces only.

3 Windows only.

4 Windows only for column only.

Methods

(1 of 2)

ADD-CALC-COLUMN( ) method	ADD-COLUMNS-FROM( ) method
ADD-LIKE-COLUMN( ) method	CLEAR-SELECTION( ) method
CREATE-RESULT-LIST-ENTRY( ) method	DELETE-CURRENT-ROW( ) method
DELETE-RESULT-LIST-ENTRY( ) method	DELETE-SELECTED-ROW( ) method
DELETE-SELECTED-ROWS( ) method	DESELECT-FOCUSED-ROW( ) method
DESELECT-ROWS( ) method	DESELECT-SELECTED-ROW( ) method
EDIT-CLEAR( ) method	EDIT-COPY( ) method
EDIT-CUT( ) method	EDIT-PASTE( ) method
EDIT-UNDO( ) method	END-FILE-DROP( ) method
FETCH-SELECTED-ROW( ) method	GET-BROWSE-COLUMN( ) method
GET-DROPPED-FILE( ) method	GET-REPOSITIONED-ROW( ) method
INSERT-ROW( ) method	IS-ROW-SELECTED( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-COLUMN( ) method

(2 of 2)

MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method
REFRESH( ) method	SCROLL-TO-CURRENT-ROW( ) method
SCROLL-TO-SELECTED-ROW( ) method	SELECT-ALL( ) method
SELECT-FOCUSED-ROW( ) method	SELECT-NEXT-ROW( ) method
SELECT-PREV-ROW( ) method	SELECT-ROW( ) method
SET-REPOSITIONED-ROW( ) method	SET-SELECTION( ) method
VALIDATE( ) method	

Events

Default keyboard events	Developer events
Field editing key function events	Mouse events
Navigation key function events	Universal key function events
DEFAULT-ACTION	DESELECTION
DROP-FILE-NOTIFY	END statement
END-MOVE[1]	END-RESIZE[1]
END-ROW-RESIZE[1]	END-SEARCH[1]
ENTRY statement	LEAVE statement
OFF-END	OFF-HOME
ROW-DISPLAY	ROW-ENTRY
ROW-LEAVE	SCROLL-NOTIFY
SELECTION	START-MOVE[1]
START-RESIZE[1]	START-ROW-RESIZE[1]
START-SEARCH[1]	VALUE-CHANGED

[1] Windows only.

See also

The chapter on the browse in *OpenEdge Development: Progress 4GL Handbook*.

BUTTON widget

A button widget represents a push button on the screen. The button can contain a textual label or it can have images associated with its pressed and unpressed states. You can define a static button with the DEFINE BUTTON statement. You can create dynamic buttons with the CREATE widget statement. This figure shows three buttons:

Attributes

(1 of 2)

AUTO-END-KEY attribute	AUTO-GO attribute	AUTO-RESIZE attribute
BGCOLOR attribute[3]	COLUMN attribute	CONTEXT-HELP-ID attribute[3,5]
CONVERT-3D-COLORS attribute[3,5]	DCOLOR attribute[4]	DEFAULT attribute[2]
DROP-TARGET attribute	DYNAMIC attribute[1]	FGCOLOR attribute[3]
FLAT-BUTTON attribute[5]	FONT attribute[3]	FRAME attribute
FRAME-COL attribute[1]	FRAME-NAME attribute[1]	FRAME-ROW attribute[1]
FRAME-X attribute[1]	FRAME-Y attribute[1]	HANDLE attribute[1]
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HELP attribute
HIDDEN attribute	HTML-CHARSET attribute[1,5]	IMAGE attribute
IMAGE-DOWN attribute	IMAGE-INSENSITIVE attribute	IMAGE-UP attribute
LABEL attribute	MANUAL-HIGHLIGHT attribute[3]	MENU-KEY attribute
MENU-MOUSE attribute[4]	MOUSE-POINTER attribute	MOVABLE attribute[3]
NAME attribute	NEXT-SIBLING attribute[1]	NEXT-TAB-ITEM attribute[1]
NO-FOCUS attribute[2,3,5]	NUM-DROPPED-FILES attribute[1]	PARENT attribute
PFCOLOR attribute[4]	POPUP-MENU attribute	PREV-SIBLING attribute[1]
PREV-TAB-ITEM attribute[1]	PRIVATE-DATA attribute	RESIZABLE attribute[3]

(2 of 2)

ROW attribute	SELECTABLE attribute[3]	SELECTED attribute
SENSITIVE attribute	TAB-POSITION attribute[1]	TAB-STOP attribute
TOOLTIP attribute[3,5]	TYPE attribute[1]	VISIBLE attribute
WIDGET-ID attribute [3,5]	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute[1]	X attribute	Y attribute

1 Readable only.

2 Can be set only before the button widget is realized.

3 Graphical interfaces only.

4 Character interfaces only.

5 Windows only.

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-IMAGE( ) method	LOAD-IMAGE-DOWN( ) method
LOAD-IMAGE-INSENSITIVE( ) method	LOAD-IMAGE-UP( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	

Events

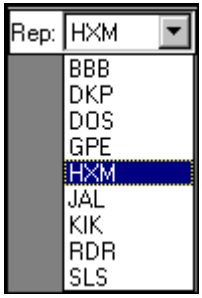
Default keyboard events	Developer events
Mouse events	Navigation key function events
Universal key function events	CHOOSE
DROP-FILE-NOTIFY	ENTRY
LEAVE	

See also

The chapter on buttons, images, and rectangles in *OpenEdge Development: Progress 4GL Handbook*.

COMBO-BOX widget

A combo box is a field-level widget that combines the functionality of a fill-in field, radio set, and selection list into one fill-in and drop down list. You can set up a static combo box widget with the VIEW-AS phrase. You can create a dynamic combo box with the CREATE widget statement.



Attributes

(1 of 2)

AUTO-COMPLETION attribute[3,5]	AUTO-RESIZE attribute	AUTO-ZAP attribute
BGCOLOR attribute[3]	COLUMN attribute	CONTEXT-HELP-ID attribute[3,5]
CURSOR-OFFSET attribute	DATA-TYPE attribute	DBNAME attribute[1]
DCOLOR attribute[4]	DELIMITER attribute	DISABLE-AUTO-ZAP attribute
DROP-TARGET attribute	DYNAMIC attribute[1]	EDIT-CAN-PASTE attribute[3,5]
EDIT-CAN-UNDO attribute[3,5]	FGCOLOR attribute[3]	FONT attribute[3]
FORMAT attribute	FRAME attribute	FRAME-COL attribute[1]
FRAME-NAME attribute[1]	FRAME-ROW attribute[1]	FRAME-X attribute[1]
FRAME-Y attribute[1]	HANDLE attribute[1]	HEIGHT-CHARS attribute[1]
HEIGHT-PIXELS attribute[1]	HELP attribute	HIDDEN attribute
HTML-CHARSET attribute[1,5]	INNER-LINES attribute[2]	INPUT-VALUE attribute

(2 of 2)

LABEL attribute	LABELS attribute[1]	LIST-ITEMS attribute
MANUAL-HIGHLIGHT attribute[3]	MAX-CHARS attribute[3]	MENU-KEY attribute
MENU-MOUSE attribute[3]	MODIFIED attribute	MOUSE-POINTER attribute
MOVABLE attribute[3]	NAME attribute	NEXT-SIBLING attribute[1]
NEXT-TAB-ITEM attribute[1]	NUM-DROPPED-FILES attribute[1]	NUM-ITEMS attribute[1]
PARENT attribute	PFCOLOR attribute[4]	POPUP-MENU attribute
PREV-SIBLING attribute[1]	PREV-TAB-ITEM attribute[1]	PRIVATE-DATA attribute
RESIZABLE attribute[3]	ROW attribute	SCREEN-VALUE attribute
SELECTABLE attribute[3]	SELECTED attribute	SELECTION-END attribute[3,5]
SELECTION-START attribute[3,5]	SELECTION-TEXT attribute[3,5]	SENSITIVE attribute
SIDE-LABEL-HANDLE attribute	SORT attribute[2]	SUBTYPE attribute[1,3,5]
TABLE attribute[1]	TAB-POSITION attribute[1]	TAB-STOP attribute
TEXT-SELECTED attribute[3,5]	TOOLTIP attribute[3,5]	TYPE attribute[1]
UNIQUE-MATCH attribute[3,5]	VISIBLE attribute	WIDGET-ID attribute [3,5]
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute[1]
X attribute	Y attribute	

[1] Readable only.

[2] Character interfaces and Windows only.

[3] Graphical interfaces only.

[4] Character interfaces only.

[5] Windows only.

Methods

ADD-FIRST( ) method	ADD-LAST( ) method
CLEAR-SELECTION( ) method	DELETE( ) method
EDIT-CLEAR( ) method	EDIT-COPY( ) method
EDIT-CUT( ) method	EDIT-PASTE( ) method
EDIT-UNDO( ) method	END-FILE-DROP( ) method
ENTRY( ) method	GET-DROPPED-FILE( ) method
INSERT( ) method	LOAD-MOUSE-POINTER( ) method
LOOKUP( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	REPLACE( ) method
SET-SELECTION( ) method	VALIDATE( ) method

Events

Default keyboard events	Developer events
Field editing key function events	General direct manipulation events
Mouse events	Navigation key function events
Universal key function events	DROP-FILE-NOTIFY
ENTRY	LEAVE
VALUE-CHANGED	

See also

The chapter on representing data in *OpenEdge Development: Progress 4GL Handbook*.

CONTROL-FRAME widget (Windows only; Graphical interfaces only)

A control-frame is a field-level widget that holds an ActiveX control that you select for your application from the OpenEdge AppBuilder. A control-frame is always created dynamically.

A control-frame has no visualization.

Progress instantiates two separate but related objects when you create a control-frame:

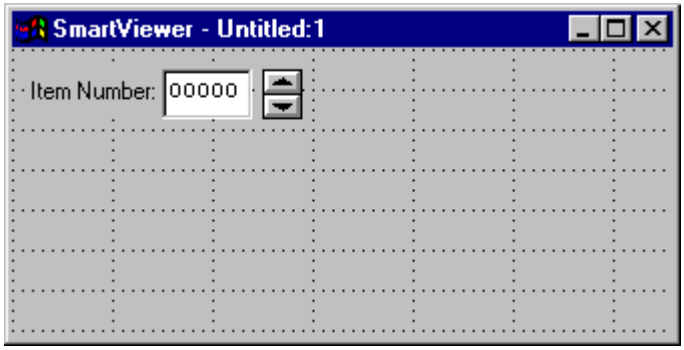
- A control-frame widget.
- A control-frame COM object.

The widget itself provides a connection between the ActiveX control and the Progress user interface. When the widget is realized, Progress creates a COM object that provides the real ActiveX control container support. Thus, the control-frame widget provides widget attributes and methods to manage the Progress side of the interface, while the control-frame COM object provides COM object properties and methods to gain access to the control itself.

When you insert an ActiveX control into your application, the AppBuilder creates a control-frame with the CREATE widget statement and specifies a default name (NAME attribute value) for the widget. The AppBuilder creates a design-time instance of the ActiveX control based on the control you select in the AppBuilder, making its design-time properties available to the AppBuilder. When you save your application, the AppBuilder saves the design-time instance in a separate file (with `.wrx` extension) for use at run time.

At run time, your application accesses the control indirectly through the control-frame widget. First, you use the COM-HANDLE widget attribute to return a component handle to the control-frame COM object. Second, you use this handle to access properties and methods of the control-frame COM object, which provide access to the ActiveX control itself.

This is a SmartViewer into which a developer, using the AppBuilder, has dropped a literal widget, a fill-in widget, and a control-frame widget. The control-frame widget holds a Crescent spin control.



Attributes

Control-frame widget attributes

(1 of 2)

BGCOLOR attribute[1]	COLUMN attribute[2]	COM-HANDLE attribute[3]
CONTEXT-HELP-ID attribute	DYNAMIC attribute[3]	FRAME attribute
FRAME-COL attribute[3]	FRAME-NAME attribute[3]	FRAME-ROW attribute[3]
FRAME-X attribute[3]	FRAME-Y attribute[3]	HEIGHT-CHARS attribute[2]
HEIGHT-PIXELS attribute[2]	HELP attribute	HIDDEN attribute
HTML-CHARSET attribute	NAME attribute[2]	NEXT-SIBLING attribute[3]
NEXT-TAB-ITEM attribute[3]	PARENT attribute	PREV-SIBLING attribute[3]
PREV-TAB-ITEM attribute[3]	PRIVATE-DATA attribute	ROW attribute[2]
SENSITIVE attribute	TAB-POSITION attribute[3]	TAB-STOP attribute

(2 of 2)

TYPE attribute[3]	VISIBLE attribute	WIDGET-ID attribute [4,5]
WIDTH-CHARS attribute[2]	WIDTH-PIXELS attribute[2]	WINDOW attribute[3]
X attribute[2]	Y attribute[2]	

1 FGCOLOR has no meaning because the ActiveX control visualization constitutes the foreground.

2 Mapped to a corresponding control-frame COM object property.

3 Readable only.

4 Graphical interfaces only.

5 Windows only.

Properties — Control-frame COM object properties[1]

Control-Name property[2]	Controls property	Height property[3]
Left property[3]	Name property[3]	Top property[3]
Widget-Handle property	Width property[3]	

1 Accessible using a component handle set to the control-frame COM-HANDLE attribute value.

2 The name of an ActiveX control that is contained by the control-frame COM object.

3 Mapped to a corresponding control-frame widget attribute.

Methods — Control-frame widget methods

ADD-EVENTS-PROCEDURE( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	REMOVE-EVENTS-PROCEDURE( ) method

Control-frame COM object methods[1]

LoadControls( ) method

[1] Accessible using a component handle set to the control-frame COM-HANDLE attribute value.

Events

Developer events	BACK-TAB navigation key function event
END-ERROR universal key function event	ENTRY
GO universal key function event	HELP universal key function event
LEAVE	TAB navigation key function event

Notes

- You must use the AppBuilder to incorporate one or more ActiveX control instances into a Progress 4GL application. The AppBuilder, operating in design mode, provides the facilities to set design-time properties for ActiveX controls.
- After incorporating ActiveX controls into an application with the AppBuilder, the resulting window file, when compiled and executed, interacts with the ActiveX controls at run time.
- To access a loaded ActiveX control at run time, use the control-frame COM-HANDLE attribute to get a handle to the control-frame COM object. To return a handle to the control, use the design-time name of the ActiveX control as a property of the control-frame COM object:

```
DEFINE VARIABLE hCFwid AS WIDGET-HANDLE. /* Control Frame widget */
DEFINE VARIABLE hCFcom AS COM-HANDLE.    /* Control Frame COM Object */
DEFINE VARIABLE hDateSpin AS COM-HANDLE. /* ActiveX Control */

/* ... Control-frame created with handle hCFwid and loaded with
        ActiveX control named DateSpin ... */

hCFcom = hCFwid:COM-HANDLE.
hDateSpin = hCFcom:DateSpin.
```

  As an alternative, use the COM object Controls property to return a handle to a control collection. Use the control collection Item(1) method call to return the handle to the ActiveX control. (This control collection object provides support for searching multiple ActiveX controls in a control-frame, available in a future release of OpenEdge.)

- You can use a single ActiveX control more than once in a single window file. Each time you insert the control, the AppBuilder creates a separate control-frame for it with a unique NAME attribute value.
- Some control-frame widget attributes correspond to control-frame COM object properties so that setting one sets the other. You must directly set and read all ActiveX control run-time properties using a handle (also a COM-HANDLE value) to the control.
- To trap control-frame events, use the ON statement, as with any 4GL widget. To trap events for the associated ActiveX control, you must use ActiveX control (OCX) event procedures. Also, to "apply" an ActiveX control event from the 4GL, run the event procedure directly, like any 4GL internal procedure. The APPLY statement has no effect on ActiveX controls. For more information, see the reference entries for the PROCEDURE statement and RUN statement.
- Progress control-frame events are mutually exclusive with associated ActiveX control events. That is, only one event handler, either an ON trigger or an event procedure, fires for a single event.

See also The chapter on ActiveX control container support in *OpenEdge Development: Programming Interfaces*.

DIALOG-BOX widget

A dialog box is a special type of frame that is displayed in its own window. A dialog box differs from a window in two major respects:

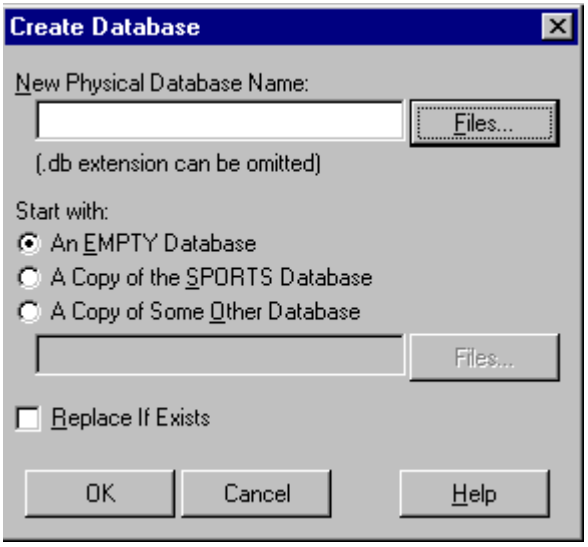
- It has a system window ventilator, but has no affordances for minimizing or maximizing.
- While a dialog box has input focus, your application cannot perform any other processing until you complete the input or otherwise close the dialog box. That is, it is modal.

You can specify that a frame be displayed as a dialog box by using the VIEW-AS phrase. You can create a dynamic dialog box with the CREATE widget statement.

A dialog box can contain a frame family acting as the root frame. However a dialog box cannot be a child of another frame or dialog box; it can only be parented by a window.

The following dialog box contains:

- Two fill-ins
- One radio set
- One toggle box
- Five buttons



Attributes

(1 of 2)

BACKGROUND attribute[1]	BGCOLOR attribute[2]	BORDER-BOTTOM-CHARS attribute[1]
BORDER-BOTTOM-PIXELS attribute[1]	BORDER-LEFT-CHARS attribute[1]	BORDER-LEFT-PIXELS attribute[1]
BORDER-RIGHT-CHARS attribute[1]	BORDER-RIGHT-PIXELS attribute[1]	BORDER-TOP-CHARS attribute[1]
BORDER-TOP-PIXELS attribute[1]	BOX-SELECTABLE attribute[2]	CANCEL-BUTTON attribute
COLUMN attribute	CONTEXT-HELP attribute[2]	CONTEXT-HELP-FILE attribute[2]
CURRENT-ITERATION attribute[1]	DCOLOR attribute[3]	DEFAULT-BUTTON attribute
DROP-TARGET attribute	DYNAMIC attribute[1]	FGCOLOR attribute[2]
FIRST-CHILD attribute[1]	FONT attribute[2]	HANDLE attribute[1]
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HIDDEN attribute
HTML-CHARSET attribute[1]	LAST-CHILD attribute[1]	MENU-KEY attribute
MENU-MOUSE attribute[3]	MOUSE-POINTER attribute	NAME attribute
NEXT-SIBLING attribute[1]	NUM-DROPPED-FILES attribute[1]	NUM-SELECTED-WIDGETS attribute[1]
PARENT attribute	PFCOLOR attribute[3]	POPUP-MENU attribute
PREV-SIBLING attribute[1]	PRIVATE-DATA attribute	ROW attribute
SCROLLABLE attribute	SENSITIVE attribute	THREE-D attribute
TITLE attribute	TITLE-BGCOLOR attribute[2]	TITLE-DCOLOR attribute[3]
TITLE-FGCOLOR attribute[2]	TITLE-FONT attribute[2]	TYPE attribute[1]
VIRTUAL-HEIGHT-CHARS attribute	VIRTUAL-HEIGHT-PIXELS attribute	VIRTUAL-WIDTH-CHARS attribute

(2 of 2)

VIRTUAL-WIDTH-PIXELS attribute	VISIBLE attribute	WIDGET-ID attribute [2,4]
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute[1]
X attribute	Y attribute	

[1] Readable only.

[2] Supported in graphical interfaces only.

[3] Supported in character interfaces only.

[4] Windows only.

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method)
GET-SELECTED-WIDGET( ) method	LOAD-MOUSE-POINTER( ) method
VALIDATE( ) method	

Events

Developer events	Frame-only direct manipulation events
Universal key function events	DROP-FILE-NOTIFY
ENTRY	LEAVE
WINDOW-CLOSE	

Note

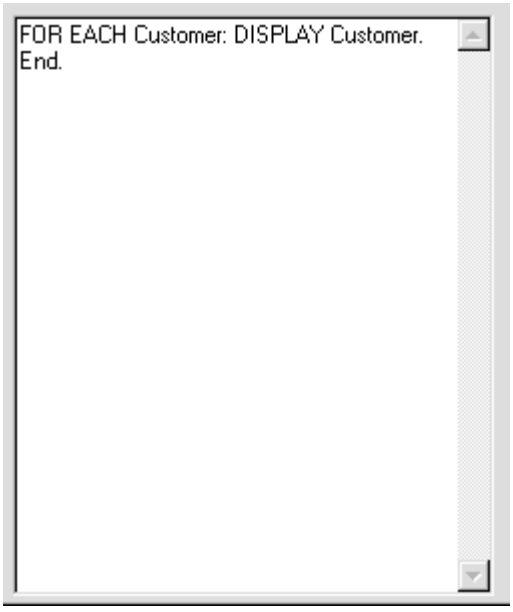
Generally, your application must wait to complete dialog box input before continuing with other processing. However, the WAIT-FOR statement for the procedure can also respond to an event for a procedure handle as long as the widget in the WAIT-FOR statement widget list is a procedure handle.

See also

FRAME widget, WAIT-FOR statement, WINDOW widget

EDITOR widget

An editor is a field-level widget that allows you to perform complex text manipulation on a character value. You can set up a static editor widget with the VIEW-AS phrase. You can create a dynamic editor widget with the CREATE widget statement.



Attributes

(1 of 3)

AUTO-INDENT attribute	AUTO-RESIZE attribute	BGCOLOR attribute[5]
BOX attribute[3, 5]	BUFFER-CHARS attribute[1,2]	BUFFER-LINES attribute[1,2]
COLUMN attribute	CONTEXT-HELP-ID attribute[3,5]	CURSOR-CHAR attribute
CURSOR-LINE attribute	CURSOR-OFFSET attribute	DATA-TYPE attribute
DBNAME attribute	DCOLOR attribute[2]	DROP-TARGET attribute
DYNAMIC attribute[4]	EDIT-CAN-PASTE attribute[3, 4, 5]	EDIT-CAN-UNDO attribute[3, 5]
EMPTY attribute[4]	FGCOLOR attribute[5]	FONT attribute[5]
FRAME attribute[4]	FRAME-COL attribute[4]	FRAME-NAME attribute[4]

(2 of 3)

FRAME-ROW attribute[4]	FRAME-X attribute[4]	FRAME-Y attribute[4]
HANDLE attribute[4]	HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
HELP attribute	HIDDEN attribute	HTML-CHARSET attribute[3, 4]
INNER-CHARS attribute	INNER-LINES attribute	INPUT-VALUE attribute
LABEL attribute	LABELS attribute[4]	LARGE attribute[1,3]
LENGTH attribute	MANUAL-HIGHLIGHT attribute[5]	MAX-CHARS attribute[1, 5]
MENU-KEY attribute	MENU-MOUSE attribute[5]	MODIFIED attribute
MOUSE-POINTER attribute	MOVABLE attribute[5]	NAME attribute
NEXT-SIBLING attribute[4]	NEXT-TAB-ITEM attribute[4]	NUM-DROPPED-FILES attribute[4]
NUM-LINES attribute	NUM-REPLACED attribute	PARENT attribute
PFCOLOR attribute[2]	POPUP-MENU attribute	PREV-SIBLING attribute[4]
PREV-TAB-ITEM attribute[4]	PRIVATE-DATA attribute	PROGRESS-SOURCE attribute[1,2]
READ-ONLY attribute	RESIZABLE attribute[5]	RETURN-INSERTED attribute[1,3]
ROW attribute	SCREEN-VALUE attribute	SCROLLBAR-HORIZONTAL attribute[1]
SCROLLBAR-VERTICAL attribute[1]	SELECTABLE attribute[5]	SELECTED attribute
SELECTION-END attribute[4]	SELECTION-START attribute[4]	SELECTION-TEXT attribute[4]
SENSITIVE attribute	SIDE-LABEL-HANDLE attribute	TABLE attribute[4]
TAB-POSITION attribute[4]	TAB-STOP attribute	TEXT-SELECTED attribute[4]
TOOLTIP attribute[3,5]	TYPE attribute[4]	VISIBLE attribute

(3 of 3)

WIDGET-ID attribute [3,5]	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute[4]	WORD-WRAP attribute[1]	X attribute
Y attribute		

[1] Can be set only before the editor widget is realized.

[2] Supported in character interfaces only.

[3] Supported in Windows only.

[4] Readable only.

[5] Supported in graphical interfaces only.

Methods

CLEAR-SELECTION( ) method	CONVERT-TO-OFFSET( ) method
DELETE-CHAR( ) method	DELETE-LINE( ) method
EDIT-CLEAR( ) method	EDIT-COPY( ) method
EDIT-CUT( ) method	EDIT-PASTE( ) method
EDIT-UNDO( ) method	END-FILE-DROP( ) method
GET-DROPPED-FILE( ) method	INSERT-BACKTAB( ) method[1]
INSERT-FILE( ) method	INSERT-STRING( ) method
INSERT-TAB( ) method[1]	LOAD-MOUSE-POINTER( ) method
MOVE-AFTER-TAB-ITEM( ) method	MOVE-BEFORE-TAB-ITEM( ) method
MOVE-TO-BOTTOM( ) method	MOVE-TO-EOF( ) method
MOVE-TO-TOP( ) method	READ-FILE( ) method
REPLACE( ) method	REPLACE-SELECTION-TEXT( ) method
SAVE-FILE( ) method	SEARCH( ) method
SET-SELECTION( ) method	VALIDATE( ) method

[1] Supported in character interfaces only.

Events

Default keyboard events	Developer events
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DROP-FILE-NOTIFY	ENTRY
LEAVE	VALUE-CHANGED

See also

The chapter on representing data in *OpenEdge Development: Progress 4GL Handbook*.

FIELD-GROUP widget

A field group is the hidden parent of field-level widgets and child frames owned by a parent frame or dialog box. Thus, field groups are the actual children of frames and dialog boxes. A frame contains the following field groups:

- A background field group (which includes the frame header).
- For a one-down frame or dialog box: A single data field group containing field-level widgets and child frames.
- For a multiple-down frame: One data field group for each data iteration in the frame.

A field group has no visible representation. You cannot explicitly define or create field groups. They are generated automatically when frames are defined or created.

Attributes

COLUMN attribute	DYNAMIC attribute	FIRST-CHILD attribute
FIRST-TAB-ITEM attribute	FOREGROUND attribute	HANDLE attribute
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HTML-CHARSET attribute[1]
LAST-CHILD attribute	LAST-TAB-ITEM attribute	NAME attribute
NEXT-SIBLING attribute	NUM-TABS attribute	PARENT attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	ROW attribute
SENSITIVE attribute	TYPE attribute[1]	VISIBLE attribute
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute[2]
X attribute	Y attribute	

[1] Windows only.

[2] Readable only.

Note: For a field group, all of these attributes are read-only except for PRIVATE-DATA and SENSITIVE.

Methods

GET-TAB-ITEM( ) method	LOAD-MOUSE-POINTER( ) method

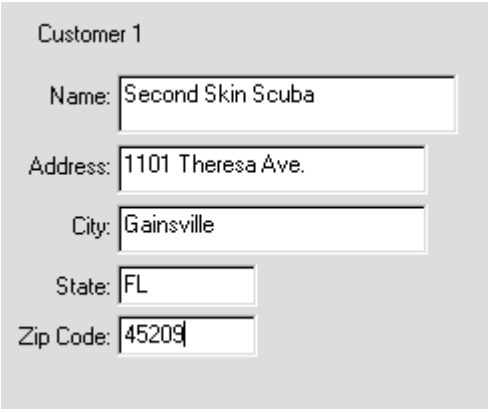
Events The FIELD-GROUP widget does not support any events.

See also DIALOG-BOX widget, FRAME widget

FILL-IN widget

A fill-in widget is the simplest form of data representation. Within a fill-in, the field value is displayed as a string of characters that you can edit. A fill-in is the default representation for data. You can explicitly set up a static fill-in with the VIEW-AS phrase. You can create a dynamic fill-in with the CREATE widget statement.

Note: The default sizing of fill-ins occurs only when you use the default font. When you explicitly specify a font, Progress uses the average width of that font.



Attributes

(1 of 2)

ATTR-SPACE attribute	AUTO-RESIZE attribute	AUTO-RETURN attribute
AUTO-ZAP attribute	BGCOLOR attribute[5]	BLANK attribute
COLUMN attribute	CONTEXT-HELP-ID attribute[3,5]	CURSOR-OFFSET attribute
DATA-TYPE attribute[1]	DBNAME attribute[4]	DCOLOR attribute[2]
DEBLANK attribute	DISABLE-AUTO-ZAP attribute	DROP-TARGET attribute
DYNAMIC attribute[4]	EDIT-CAN-PASTE attribute[3, 4,5]	EDIT-CAN-UNDO attribute[3, 5]
FGCOLOR attribute[5]	FONT attribute[5]	FORMAT attribute
FRAME attribute	FRAME-COL attribute[4]	FRAME-NAME attribute[4]
FRAME-ROW attribute[4]	FRAME-X attribute[4]	FRAME-Y attribute[4]

(2 of 2)

HANDLE attribute[4]	HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
HELP attribute	HIDDEN attribute	HTML-CHARSET attribute[3]
INDEX attribute	INPUT-VALUE attribute	LABEL attribute
LABELS attribute[4]	MANUAL-HIGHLIGHT attribute[5]	MENU-KEY attribute
MENU-MOUSE attribute[5]	MODIFIED attribute	MOUSE-POINTER attribute
MOVABLE attribute[5]	NAME attribute	NEXT-SIBLING attribute[4]
NEXT-TAB-ITEM attribute[4]	NUM-DROPPED-FILES attribute[4]	PARENT attribute
PASSWORD-FIELD attribute	PASSWORD-FIELD attribute	PFCOLOR attribute[2]
POPUP-MENU attribute	PREV-SIBLING attribute[4]	PREV-TAB-ITEM attribute[4]
PRIVATE-DATA attribute	READ-ONLY attribute	RESIZABLE attribute[5]
ROW attribute	SCREEN-VALUE attribute	SELECTABLE attribute[5]
SELECTED attribute	SELECTION-END attribute	SELECTION-START attribute
SELECTION-TEXT attribute	SENSITIVE attribute	SIDE-LABEL-HANDLE attribute
SUBTYPE attribute[1]	TABLE attribute[4]	TAB-POSITION attribute[4]
TAB-STOP attribute	TEXT-SELECTED attribute	TOOLTIP attribute[3,5]
TYPE attribute[4]	VISIBLE attribute	WIDGET-ID attribute [3,5]
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute[4]
X attribute	Y attribute	

1 Can be set only before the fill-in widget is realized.

2 Character interfaces only.

3 Windows only.

4 Readable only.

5 Graphical interfaces only.

Methods

CLEAR-SELECTION( ) method	EDIT-CLEAR( ) method
EDIT-COPY( ) method	EDIT-CUT( ) method
EDIT-PASTE( ) method	EDIT-UNDO( ) method
END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	SET-SELECTION( ) method
VALIDATE( ) method	

Events

Default keyboard events	Developer events
Field editing key function events	General direct manipulation events
Mouse events	Navigation key function events
Universal key function events	DROP-FILE-NOTIFY
ENTRY	LEAVE
VALUE-CHANGED	

See also

The chapter on representing data in *OpenEdge Development: Progress 4GL Handbook*.

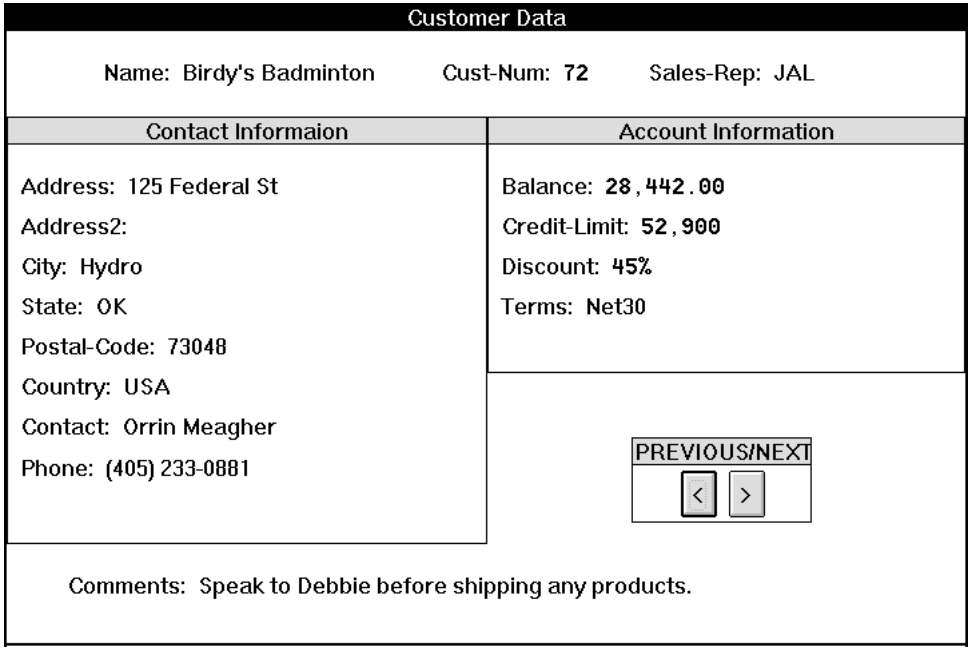
FRAME widget

A frame is a display area within a window that can group together (contain) a set of field-level widgets and child frames. In addition to default frames set up by Progress, you can set up static frames with the FRAME phrase or DEFINE FRAME statement. You can create a dynamic one-down frame with the CREATE WIDGET statement.

Related field-level widgets and child frames are actually parented by a single field group widget, which is owned, in turn, by the parenting frame. You parent static field-level widgets to a static frame using a DEFINE FRAME, FORM, or FRAME I/O statement. You parent dynamic field-level widgets to any frame by setting the FRAME attribute of each field-level widget to the handle of the parent frame. You can parent frame widgets to any frame by setting the FRAME attribute of each child frame to the handle of its parent frame.

Frames in a parent and child relationship form a frame family, which is a hierarchy of parent and child frames ultimately parented by a window. The top parent frame that is parented by the window is the root frame of the frame family.

The following figure shows a frame family with four frames, including three child frames titled Contact Information, Account Information, and PREVIOUS/NEXT:



Attributes

(1 of 2)

BACKGROUND attribute[6]	BGCOLOR attribute[5]	BLOCK-ITERATION-DISPLAY attribute[6]
BORDER-BOTTOM-CHARS attribute[6]	BORDER-BOTTOM-PIXELS attribute[6]	BORDER-LEFT-CHARS attribute[6]
BORDER-LEFT-PIXELS attribute[6]	BORDER-RIGHT-CHARS attribute[6]	BORDER-RIGHT-PIXELS attribute[6]
BORDER-TOP-CHARS attribute[6]	BORDER-TOP-PIXELS attribute[6]	BOX attribute[1]
BOX-SELECTABLE attribute[5]	CANCEL-BUTTON attribute	CAREFUL-PAINT attribute
CENTERED attribute	COLUMN attribute	FRAME attribute[1]
GRID-FACTOR-HORIZONTAL attribute[5]	GRID-FACTOR-VERTICAL attribute[5]	GRID-SNAP attribute[5]
GRID-UNIT-HEIGHT-CHARS attribute[5]	GRID-UNIT-HEIGHT-PIXELS attribute[5]	GRID-UNIT-WIDTH-CHARS attribute[5]
GRID-UNIT-WIDTH-PIXELS attribute[5]	GRID-VISIBLE attribute[5]	HANDLE attribute[6]
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HIDDEN attribute
HTML-CHARSET attribute[2, 6]	LABELS attribute[6]	LAST-CHILD attribute[6]
LINE attribute[6]	MANUAL-HIGHLIGHT attribute[5]	PARENT attribute[1]
PFCOLOR attribute[4]	POPUP-MENU attribute	PREV-SIBLING attribute[6]
PREV-TAB-ITEM attribute	PRIVATE-DATA attribute	RESIZABLE attribute[5]
ROW attribute	SCROLLABLE attribute	SELECTABLE attribute[5]
SELECTED attribute	SENSITIVE attribute	SIDE-LABELS attribute
TAB-POSITION attribute[6]	TAB-STOP attribute	THREE-D attribute[2]

(2 of 2)

TITLE attribute[1,3]	TITLE-BGCOLOR attribute[5]	TITLE-DCOLOR attribute[4]
TITLE-FGCOLOR attribute[5]	TITLE-FONT attribute[5]	TOP-ONLY attribute
WIDGET-ID attribute [2, 5]		

1 Can be set only before the frame widget is realized.

2 Windows only.

3 If the frame does not have a title when Progress realizes it, you cannot add one, but you can change the existing title.

4 Character interfaces only.

5 Graphical interfaces only.

6 Readable only.

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
GET-INDEX-BY-NAMESPACE-NAME( ) method	GET-SELECTED-WIDGET( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	VALIDATE( ) method

Events

Developer events	Frame-only direct manipulation events
General direct manipulation events	Mouse events
Universal key function events	DDE-NOTIFY[1]
DROP-FILE-NOTIFY	ENTRY
LEAVE	

1 Windows only. This event occurs only in dynamic data exchange (DDE) conversations. For more information, see the chapter on DDE in *OpenEdge Development: Programming Interfaces*.

Notes

- Field-level widgets and child frames are not directly parented by a parent frame. They are parented by field groups that are owned by the parent frame. Thus, you can also parent a child frame by setting the child frame's PARENT attribute to the widget handle of a field group in the parent frame.

  To access all the field-level widgets and child frames owned by a frame, you must first use the frame's FIRST-CHILD or LAST-CHILD attribute to find a field group within the frame. You can then use the field group's NEXT-SIBLING or PREV-SIBLING attribute to find other field groups in the frame. You can use the field group's FIRST-CHILD or LAST-CHILD attribute to find a field-level widget or child frame within the field group. You can then use the field-level widget's or child frame's NEXT-SIBLING or PREV-SIBLING attribute to find other field-level widgets and child frames within the frame.

- Child frames do not inherit the attributes of a parent frame.

- When any of a frame's field-level widgets or child frames are viewed using the DISPLAY or ENABLE statement, the parent frame also becomes visible unless its HIDDEN attribute or the HIDDEN attribute of an ancestor widget is TRUE. However, explicitly setting the VISIBLE attribute to TRUE (using the VIEW statement) for a child frame or field-level widget makes all ancestor frames visible, unless the parent or an ancestor window has its HIDDEN attribute set to TRUE.

- Child frames participate in the tab order along with any field-level widgets in the same parent frame. This means that the tab orders of all field-level widgets within a child frame is placed as a group within the tab order of the siblings of that child frame. Thus, tabbing proceeds between the field-level widgets of a root frame and the field-level widgets of all descendant frames. However, tabbing is not supported between sibling root frames (frames parented by a window). For more information on tabbing within frame families, see the chapter on frames in *OpenEdge Development: Progress 4GL Handbook*.

- You specify the position of a child frame relative to the display area of the parent frame. You must specify the position so that the upper left corner of the child frame lies within the display region of the parent frame. Otherwise at run time, when the procedure tries to realize the frame, Progress raises the ERROR condition.

- When you apply a NEXT-FRAME or PREV-FRAME navigation key function to a field-level widget, focus changes from the current frame family to the next or previous frame family (respectively) parented by the same window. That is, these key functions change focus between root frames, not between descendant frames.
- In character interfaces, the SCROLL-MODE function key is available for a frame only if the SCROLLABLE attribute of the frame is TRUE. Scroll mode allows you to use the CURSOR-RIGHT and CURSOR-LEFT keys to scroll the frame horizontally. The SCROLL-MODE function key toggles scroll mode on and off for a frame that has focus.

See also DIALOG-BOX widget, DEFINE FRAME statement, Frame phrase

IMAGE widget (Graphical interfaces only)

An image is a graphic taken from an operating system file. It can be used by itself or within a button. You can define a static image with the DEFINE IMAGE statement, and create a dynamic image with the CREATE widget statement. You can specify an image for a button using the DEFINE BUTTON statement or the button methods for loading images.

Attributes

(1 of 2)

BGCOLOR attribute[1]	COLUMN attribute	CONVERT-3D-COLORS attribute[1,3,4]
DYNAMIC attribute[2]	FGCOLOR attribute[1]	FRAME attribute
FRAME-COL attribute[2]	FRAME-NAME attribute[2]	FRAME-ROW attribute[2]
FRAME-X attribute[2]	FRAME-Y attribute[2]	HANDLE attribute[2]
HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute	HELP attribute
HIDDEN attribute	HTML-CHARSET attribute[2, 3]	IMAGE attribute
MANUAL-HIGHLIGHT attribute[1]	MOVABLE attribute[1]	NAME attribute
NEXT-SIBLING attribute[2]	PARENT attribute	PREV-SIBLING attribute[2]
PRIVATE-DATA attribute	RESIZABLE attribute[1]	RETAIN-SHAPE attribute
ROW attribute	SELECTABLE attribute[1]	SELECTED attribute
SENSITIVE attribute	STRETCH-TO-FIT attribute	TOOLTIP attribute[1,3]

(2 of 2)

TRANSPARENT attribute	TYPE attribute[2]	VISIBLE attribute
WIDGET-ID attribute [1,3]	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute[2]	X attribute	Y attribute

1 Graphical interfaces only.

2 Readable only.

3 Windows only.

4 The CONVERT-3D-COLORS attribute can be set after an image is realized, but it will not take effect until an image is loaded using the LOAD-IMAGE() method.

Methods

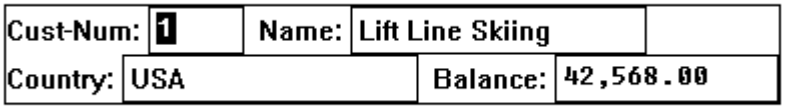
LOAD-IMAGE( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	

Events

Developer events	General direct manipulation events
Mouse events	

LITERAL widget

A literal widget is the label for a static field. If a field has a side label, you can find the handle of a literal widget by reading the field's SIDE-LABEL-HANDLE attribute. If the field has a column label, you can find the handle of the literal by examining the children of the frame's background field group. You cannot create a literal widget dynamically.



Attributes

BGCOLOR attribute[1]	COLUMN attribute	DCOLOR attribute[2]
DYNAMIC attribute[3]	FGCOLOR attribute[1]	FONT attribute[1]
FRAME attribute	FRAME-COL attribute[3]	FRAME-NAME attribute[3]
FRAME-ROW attribute[3]	FRAME-X attribute[3]	FRAME-Y attribute[3]
HANDLE attribute[3]	HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
HIDDEN attribute	HTML-CHARSET attribute[3, 4]	INPUT-VALUE attribute
MANUAL-HIGHLIGHT attribute[1]	MOVABLE attribute[1]	NAME attribute
NEXT-SIBLING attribute[3]	PARENT attribute	PREV-SIBLING attribute[3]
PRIVATE-DATA attribute	RESIZABLE attribute[1]	ROW attribute
SCREEN-VALUE attribute	SELECTABLE attribute[1]	SELECTED attribute
SENSITIVE attribute	TYPE attribute[3]	VISIBLE attribute
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute[3]
X attribute	Y attribute	

[1] Graphical interfaces only.

[2] Character interfaces only.

[3] Readable only.

[4] Windows only.

Methods

MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method

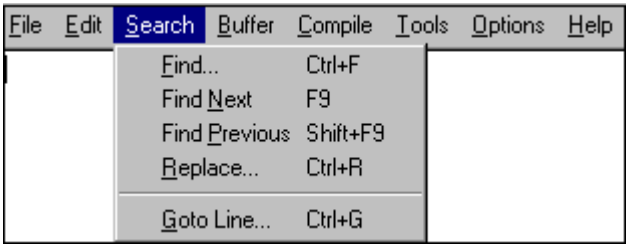
Events

The literal widget does not support any events.

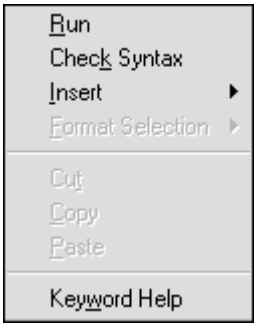
MENU widget

A menu can be a menu bar or a pop-up menu. Menu bars contain sub-menus (specifically, pull-down menus) and in some environments menu items. Pop-up menus contain menu items and sub-menus. You can define a static menu with the DEFINE MENU statement. You can create a dynamic menu with the CREATE widget statement.

The following is a menu bar:



The following is a pop-up menu:



Attributes

DCOLOR attribute[2]	DYNAMIC attribute[3]	FIRST-CHILD attribute[3]
HANDLE attribute[3]	HTML-CHARSET attribute[3, 4]	LAST-CHILD attribute[3]
NAME attribute	OWNER attribute	PFCOLOR attribute[2]
POPUP-ONLY attribute[1]	PRIVATE-DATA attribute	SENSITIVE attribute
TITLE attribute[5]	TITLE-DCOLOR attribute[2]	TYPE attribute[3]
VISIBLE attribute	WINDOW attribute[3]	

1 Graphical interfaces only.

2 Character interfaces only.

3 Readable only.

4 Windows only.

5 Can be set only before the menu widget is realized.

Note: Color and font attributes for a menu are ignored in Windows.

Methods

The MENU widget does not support any methods.

Events

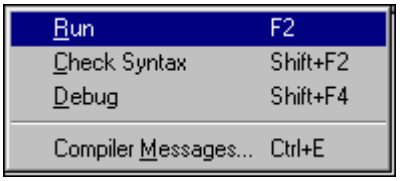
Developer events	MENU-DROP[1]

1 Supported only when the POPUP-ONLY attribute is set to TRUE and the menu is set as a popup for some other widget.

MENU-ITEM widget

A menu item is an item within a menu or submenu. A menu item can be a rule, a space, or a normal menu item. A normal menu item can be a command or a toggle-box item. Most menu item attributes and all menu item events apply only to normal menu items. You can set up a static menu item within a DEFINE MENU or DEFINE SUB-MENU statement. You can create a dynamic menu item with the CREATE widget statement.

The following is a menu containing four menu items:



Attributes

ACCELERATOR attribute	CHECKED attribute[2]	DCOLOR attribute[3]
DYNAMIC attribute[4]	HANDLE attribute[4]	HTML-CHARSET attribute[4, 5]
LABEL attribute	NAME attribute	NEXT-SIBLING attribute[4]
PARENT attribute	PFCOLOR attribute[3]	PREV-SIBLING attribute[4]
PRIVATE-DATA attribute	READ-ONLY attribute[6]	SENSITIVE attribute
SUBTYPE attribute[6]	TOGGLE-BOX attribute[6]	TYPE attribute[4]
VISIBLE attribute	WINDOW attribute[4]	

[1] Graphical interfaces only.

[2] Applies to toggle-box items only.

[3] Character interfaces only.

[4] Readable only.

[5] Windows only.

[6] Can be set only before the menu-item widget is realized.

Note: Color and font attributes for a menu item are ignored in Windows.

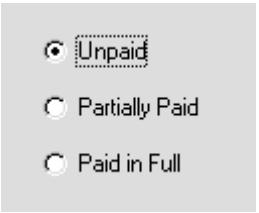
Methods The MENU-ITEM widget does not support any methods.

Events

Developer events	
VALUE-CHANGED (for toggle-box items only)	CHOOSE (except for toggle-box items)

RADIO-SET widget

A radio set is a group of values of which only one can be set at any time. You can define a static radio set by using the VIEW-AS phrase with any LOGICAL, CHARACTER, INTEGER, DECIMAL, or DATE value. You can create a dynamic radio set with the CREATE widget statement.



Attributes

(1 of 2)

AUTO-RESIZE attribute	BGCOLOR attribute[1]	COLUMN attribute
CONTEXT-HELP-ID attribute[1,5]	DATA-TYPE attribute	DBNAME attribute[2]
DCOLOR attribute[3]	DELIMITER attribute	DROP-TARGET attribute
DYNAMIC attribute[2]	EXPAND attribute[4]	FGCOLOR attribute[1]
FONT attribute[1]	FRAME attribute	FRAME-COL attribute[2]
FRAME-NAME attribute[2]	FRAME-ROW attribute[2]	FRAME-X attribute[2]
FRAME-Y attribute[2]	HANDLE attribute[2]	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
HORIZONTAL attribute[4]	HTML-CHARSET attribute[2,5]	INPUT-VALUE attribute
LABEL attribute	MANUAL-HIGHLIGHT attribute[1]	MENU-KEY attribute
MENU-MOUSE attribute[1]	MODIFIED attribute	MOUSE-POINTER attribute
MOVABLE attribute[1]	NAME attribute	NEXT-SIBLING attribute[2]
NEXT-TAB-ITEM attribute[2]	NUM-BUTTONS attribute[2]	NUM-DROPPED-FILES attribute[2]
PARENT attribute	PFCOLOR attribute[3]	POPUP-MENU attribute

(2 of 2)

PREV-SIBLING attribute[2]	PREV-TAB-ITEM attribute[2]	PRIVATE-DATA attribute
RADIO-BUTTONS attribute	RESIZABLE attribute[1]	ROW attribute
SCREEN-VALUE attribute	SELECTABLE attribute[1]	SELECTED attribute
SENSITIVE attribute	SIDE-LABEL-HANDLE attribute	TABLE attribute[2]
TAB-POSITION attribute[2]	TAB-STOP attribute	TOOLTIP attribute[1,5]
TYPE attribute[2]	VISIBLE attribute	WIDGET-ID attribute [1,5]
WIDTH-CHARS attribute	WIDTH-PIXELS attribute	WINDOW attribute[2]
X attribute	Y attribute	

1 Supported for graphical interfaces only.

2 Readable only.

3 Supported for character interfaces only.

4 Can be set only before the radio-set widget is realized.

5 Supported in Windows only.

Methods

ADD-LAST( ) method	DELETE( ) method
DISABLE( ) method	ENABLE( ) method
END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	REPLACE( ) method
VALIDATE( ) method	

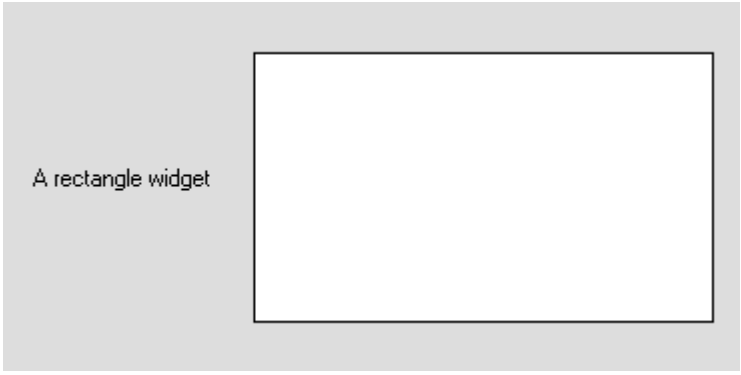
Events

Default keyboard events	Developer events
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DROP-FILE-NOTIFY	ENTRY
LEAVE	

See also The chapter on representing data in *OpenEdge Development: Progress 4GL Handbook*.

RECTANGLE widget

A rectangle is a graphical widget that can be displayed in a frame foreground or background. You can define a static rectangle with the DEFINE RECTANGLE statement. You can create a dynamic rectangle with the CREATE widget statement.



Attributes

(1 of 2)

BGCOLOR attribute[1]	COLUMN attribute	DATA-TYPE attribute
DCOLOR attribute[2]	DYNAMIC attribute[2]	EDGE-CHARS attribute
EDGE-PIXELS attribute	FGCOLOR attribute[1]	FILLED attribute
FRAME attribute	FRAME-COL attribute[2]	FRAME-NAME attribute[2]
FRAME-ROW attribute[2]	FRAME-X attribute[2]	FRAME-Y attribute[2]
GRAPHIC-EDGE attribute[3]	HANDLE attribute[2]	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
HTML-CHARSET attribute[2,4]	MANUAL-HIGHLIGHT attribute[1]	MOVABLE attribute[1]

(2 of 2)

NAME attribute	NEXT-SIBLING attribute[2]	PARENT attribute
PFCOLOR attribute[3]	PREV-SIBLING attribute[2]	PRIVATE-DATA attribute
RESIZABLE attribute[1]	ROW attribute	SELECTABLE attribute[1]
SELECTED attribute	SENSITIVE attribute	TABLE attribute[2]
TOOLTIP attribute[1,4]	TYPE attribute[2]	VISIBLE attribute
WIDGET-ID attribute [1,4]	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute[2]	X attribute	Y attribute

[1] Graphical interfaces only.

[2] Readable only.

[3] Character interfaces only.

[4] Windows only.

Methods

LOAD-MOUSE-POINTER( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	

Events

Developer events	General direct manipulation events
Mouse events	

See also

The chapter on buttons, images, and rectangles in *OpenEdge Development: Progress 4GL Handbook*.

SELECTION-LIST widget

A selection list is a widget that contains a list of possible values for a field or variable. You can use the VIEW-AS phrase to set up a static selection list. You can use the CREATE widget statement to create a dynamic selection list.

Attributes

(1 of 2)

AUTO-RESIZE attribute	BGCOLOR attribute[1]	COLUMN attribute
CONTEXT-HELP-ID attribute[1,5]	DATA-TYPE attribute	DBNAME attribute[2]
DCOLOR attribute[3]	DELIMITER attribute	DRAG-ENABLED attribute[4]
DROP-TARGET attribute	DYNAMIC attribute[2]	FGCOLOR attribute[1]
FONT attribute[1]	FRAME attribute	FRAME-COL attribute[2]
FRAME-NAME attribute[2]	FRAME-ROW attribute[2]	FRAME-X attribute[2]
FRAME-Y attribute[2]	HANDLE attribute[2]	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
HTML-CHARSET attribute[2,5]	INNER-CHARS attribute	INNER-LINES attribute
INPUT-VALUE attribute	LABEL attribute	LIST-ITEMS attribute
MANUAL-HIGHLIGHT attribute[1]	MENU-KEY attribute	MENU-MOUSE attribute[1]
MODIFIED attribute	MOUSE-POINTER attribute	MOVABLE attribute[1]
MULTIPLE attribute[2]	NAME attribute	NEXT-SIBLING attribute[2]

(2 of 2)

NEXT-TAB-ITEM attribute[2]	NUM-DROPPED-FILES attribute[2]	NUM-ITEMS attribute[2]
PARENT attribute	PFCOLOR attribute[3]	POPUP-MENU attribute
PREV-SIBLING attribute[2]	PREV-TAB-ITEM attribute[2]	PRIVATE-DATA attribute
RESIZABLE attribute[1]	ROW attribute	SCREEN-VALUE attribute
SCROLLBAR-HORIZONTAL attribute[4]	SCROLLBAR-VERTICAL attribute[4]	SELECTABLE attribute[1]
SELECTED attribute	SENSITIVE attribute	SIDE-LABEL-HANDLE attribute
SORT attribute	TABLE attribute[2]	TAB-POSITION attribute[2]
TAB-STOP attribute	TOOLTIP attribute[1,5]	TYPE attribute[2]
VISIBLE attribute	WIDGET-ID attribute [1,5]	WIDTH-CHARS attribute
WIDTH-PIXELS attribute	WINDOW attribute[2]	X attribute
Y attribute		

[1] Graphical interfaces only.

[2] Readable only.

[3] Character interfaces only.

[4] Can be set only before the selection list is realized.

[5] Windows only.

Methods

ADD-FIRST( ) method	ADD-LAST( ) method
DELETE( ) method)	END-FILE-DROP( ) method
ENTRY( ) method	GET-DROPPED-FILE( ) method
INSERT( ) method	IS-SELECTED( ) method
LOAD-MOUSE-POINTER( ) method	LOOKUP( ) method
MOVE-AFTER-TAB-ITEM( ) method	MOVE-BEFORE-TAB-ITEM( ) method)
MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method
REPLACE( ) method	SCROLL-TO-ITEM( ) method
VALIDATE( ) method	

Events

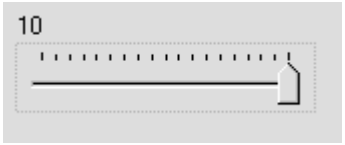
Default keyboard events	Developer events
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DEFAULT-ACTION	DROP-FILE-NOTIFY
ENTRY	LEAVE
VALUE-CHANGED	

See also

The chapter on representing data in *OpenEdge Development: Progress 4GL Handbook*.

SLIDER widget

The slider widget represents an integer value as a point on a sliding scale. You can use the VIEW-AS phrase to set up a static slider. You can use the CREATE widget statement to create a dynamic slider.



Attributes

(1 of 2)

AUTO-RESIZE attribute	BGCOLOR attribute[1]	COLUMN attribute
CONTEXT-HELP-ID attribute[1,5]	DATA-TYPE attribute	DBNAME attribute[2]
DCOLOR attribute[3]	DROP-TARGET attribute	DYNAMIC attribute[2]
FGCOLOR attribute[1]	FONT attribute[1]	FRAME attribute
FRAME-COL attribute[2]	FRAME-NAME attribute[2]	FRAME-ROW attribute[2]
FRAME-X attribute[2]	FRAME-Y attribute[2]	FREQUENCY attribute[1,5]
HANDLE attribute[2]	HEIGHT-CHARS attribute	HEIGHT-PIXELS attribute
HELP attribute	HIDDEN attribute	HORIZONTAL attribute[4]
HTML-CHARSET attribute[2,5]	INPUT-VALUE attribute	LABEL attribute
LARGE-TO-SMALL attribute	MANUAL-HIGHLIGHT attribute[1]	MAX-VALUE attribute[4]
MENU-KEY attribute	MENU-MOUSE attribute[1]	MIN-VALUE attribute[4]
MODIFIED attribute	MOUSE-POINTER attribute	MOVABLE attribute[1]
NAME attribute	NEXT-SIBLING attribute[2]	NEXT-TAB-ITEM attribute[2]
NO-CURRENT-VALUE attribute[4]	NUM-DROPPED-FILES attribute[2]	PARENT attribute
PFCOLOR attribute[3]	POPUP-MENU attribute	PREV-SIBLING attribute[2]
PREV-TAB-ITEM attribute[2]	PRIVATE-DATA attribute	RESIZABLE attribute[1]

(2 of 2)

ROW attribute	SCREEN-VALUE attribute	SELECTABLE attribute[1]
SELECTED attribute	SENSITIVE attribute	SIDE-LABEL-HANDLE attribute
TABLE attribute[2]	TAB-POSITION attribute[2]	TAB-STOP attribute
TIC-MARKS attribute[1,4,5]	TOOLTIP attribute[1,5]	TYPE attribute[2]
VISIBLE attribute	WIDGET-ID attribute [1,5]	WIDTH-CHARS attribute
WIDTH-PIXELS attribute	WINDOW attribute[2]	X attribute
Y attribute		

1 Graphical interfaces only.

2 Readable only.

3 Character interfaces only.

4 Can be set only before the slider widget is realized.

5 Windows only.

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	VALIDATE( ) method

Events

Default keyboard events	Developer events
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DROP-FILE-NOTIFY	ENTRY
LEAVE	VALUE-CHANGED

Notes

- Only a value of the INTEGER data type can be viewed as a slider.
- In character interfaces, a slider widget has a minimum width that is dependent on the specified maximum value (MAX-VALUE attribute). The minimum height for a slider widget in a character interface is 2 character units. You can specify a value as low as 1.5 character units for the height of a slider in a character interface; however, Progress rounds the value up to 2 character units.

See also

The chapter on representing data in *OpenEdge Development: Progress 4GL Handbook*.

SUB-MENU widget

A submenu can be a pull-down menu within a menu bar, or a submenu within a pull-down menu or pop-up menu. You can define a static submenu with the DEFINE SUB-MENU statement. You can use the CREATE widget statement to create a dynamic submenu.

Attributes

BGCOLOR attribute[4]	DCOLOR attribute[2]	DYNAMIC attribute[3]
FGCOLOR attribute[1]	FIRST-CHILD attribute[3]	FONT attribute[1]
HANDLE attribute[3]	HTML-CHARSET attribute[3, 4]	LABEL attribute
LAST-CHILD attribute[3]	NAME attribute	NEXT-SIBLING attribute[3]
PARENT attribute	PFCOLOR attribute[2]	PREV-SIBLING attribute[3]
PRIVATE-DATA attribute	SENSITIVE attribute	TYPE attribute[3]
VISIBLE attribute	WINDOW attribute[3]	

1 Graphical interfaces only.

2 Character interfaces only.

3 Readable only.

4 Windows only.

5 Writable only before the sub-menu widget is realized.

Note: Color and font attributes for a submenu are ignored in Windows.

Methods The SUB-MENU widget does not support any methods.

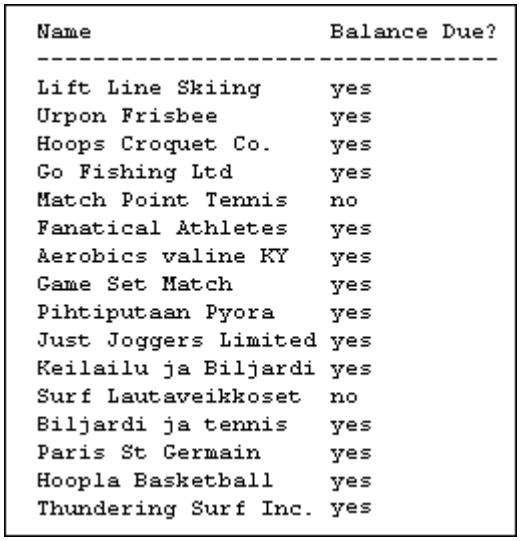
Events

Developer events	MENU-DROP

See also The chapter on menus in *OpenEdge Development: Progress 4GL Handbook*.

TEXT widget

You can use the text widget to display read-only text in a compact format. This is especially useful when you are creating hard-copy reports. You can use the VIEW-AS phrase to set up a static text widget. You can use the CREATE widget statement to create dynamic text widgets.



Name	Balance Due?
Lift Line Skiing	yes
Urpon Frisbee	yes
Hoops Croquet Co.	yes
Go Fishing Ltd	yes
Match Point Tennis	no
Fanatical Athletes	yes
Aerobics valine KY	yes
Game Set Match	yes
Pihtiputaan Pyora	yes
Just Joggers Limited	yes
Keilailu ja Biljardi	yes
Surf Lautaveikkoset	no
Biljardi ja tennis	yes
Paris St Germain	yes
Hoopla Basketball	yes
Thundering Surf Inc.	yes

Attributes

(1 of 2)

ATTR-SPACE attribute	AUTO-RESIZE attribute	BGCOLOR attribute[1]
BLANK attribute	COLUMN attribute	DATA-TYPE attribute
DBNAME attribute[2]	DCOLOR attribute[3]	DEBLANK attribute
DYNAMIC attribute[3]	FGCOLOR attribute[1]	FONT attribute[1]
FORMAT attribute	FRAME attribute	FRAME-COL attribute[3]
FRAME-NAME attribute[3]	FRAME-ROW attribute[3]	FRAME-X attribute[3]
FRAME-Y attribute[3]	HANDLE attribute[3]	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
HTML-CHARSET attribute[3,4]	INDEX attribute[3]	INPUT-VALUE attribute
LABEL attribute	LABELS attribute[2]	MANUAL-HIGHLIGHT attribute

(2 of 2)

MODIFIED attribute	MOVABLE attribute[1]	NAME attribute
NEXT-SIBLING attribute[3]	PARENT attribute	PREV-SIBLING attribute[3]
PRIVATE-DATA attribute	RESIZABLE attribute[1]	ROW attribute
SCREEN-VALUE attribute	SELECTABLE attribute[1]	SELECTED attribute
SENSITIVE attribute	SIDE-LABEL-HANDLE attribute	TABLE attribute[3]
TOOLTIP attribute[1,4]	TYPE attribute[3]	VISIBLE attribute
WIDGET-ID attribute [1,4]	WIDTH-CHARS attribute	WIDTH-PIXELS attribute
WINDOW attribute[2]	X attribute	Y attribute

[1] Graphical interfaces only.

[2] Readable only.

[3] Character interfaces only.

[4] Windows only.

Methods

MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method

Events

Developer events	General direct manipulation events
Mouse events	

Note

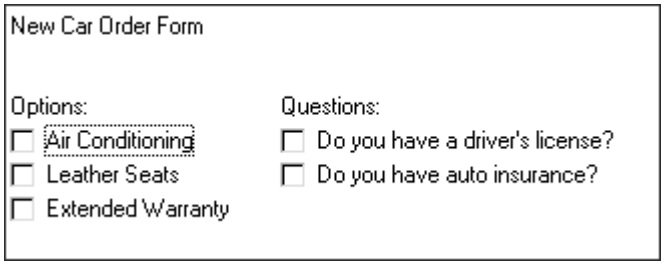
You can view a field as text by specifying VIEW-AS TEXT for the field. You can make text the default representation for all fields in a frame by specifying USE-TEXT for the frame.

See also

The chapter on representing data in *OpenEdge Development: Progress 4GL Handbook*.

TOGGLE-BOX widget

You can use the toggle box widget to represent a logical value. You can use the VIEW-AS phrase to set up a static toggle box, or the CREATE widget statement to create a dynamic toggle box. This figure shows five toggle boxes.



Attributes

(1 of 2)

AUTO-RESIZE attribute	BGCOLOR attribute[1]	CHECKED attribute
COLUMN attribute	CONTEXT-HELP-ID attribute[1,4]	DATA-TYPE attribute
DBNAME attribute[2]	DCOLOR attribute[3]	DROP-TARGET attribute
DYNAMIC attribute[2]	FGCOLOR attribute[1]	FONT attribute[1]
FORMAT attribute	FRAME attribute	FRAME-COL attribute[2]
FRAME-NAME attribute[2]	FRAME-ROW attribute[2]	FRAME-X attribute[2]
FRAME-Y attribute[2]	HANDLE attribute[2]	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HELP attribute	HIDDEN attribute
HTML-CHARSET attribute[2,4]	INPUT-VALUE attribute	LABEL attribute
MANUAL-HIGHLIGHT attribute[1]	MENU-KEY attribute	MENU-MOUSE attribute[1]
MODIFIED attribute	MOUSE-POINTER attribute	MOVABLE attribute[1]
NAME attribute	NEXT-SIBLING attribute[2]	NEXT-TAB-ITEM attribute[2]
NUM-DROPPED-FILES attribute[2]	PARENT attribute	PFCOLOR attribute[3]
POPUP-MENU attribute	PREV-SIBLING attribute[2]	PREV-TAB-ITEM attribute[2]

(2 of 2)

PRIVATE-DATA attribute	RESIZABLE attribute[1]	ROW attribute
SCREEN-VALUE attribute	SELECTABLE attribute[1]	SELECTED attribute
SENSITIVE attribute	TABLE attribute[2]	TAB-POSITION attribute[2]
TAB-STOP attribute	TOOLTIP attribute[1,4]	TYPE attribute[2]
VISIBLE attribute	WIDGET-ID attribute [1,4]	WIDTH-CHARS attribute
WIDTH-PIXELS attribute	WINDOW attribute[2]	X attribute
Y attribute		

1 Graphical interfaces only.

2 Readable only.

3 Character interfaces only.

4 Windows only.

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
LOAD-MOUSE-POINTER( ) method	MOVE-AFTER-TAB-ITEM( ) method
MOVE-BEFORE-TAB-ITEM( ) method	MOVE-TO-BOTTOM( ) method
MOVE-TO-TOP( ) method	VALIDATE( ) method

Events

Default keyboard events	Developer events
General direct manipulation events	Mouse events
Navigation key function events	Universal key function events
DROP-FILE-NOTIFY	ENTRY
LEAVE	VALUE-CHANGED

See also

The chapter on representing data in *OpenEdge Development: Progress 4GL Handbook*.

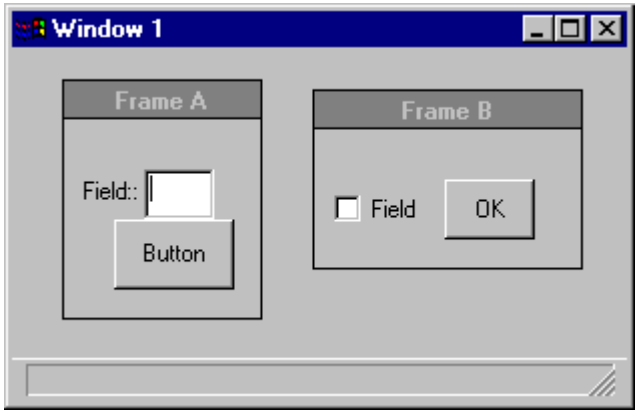
WINDOW widget

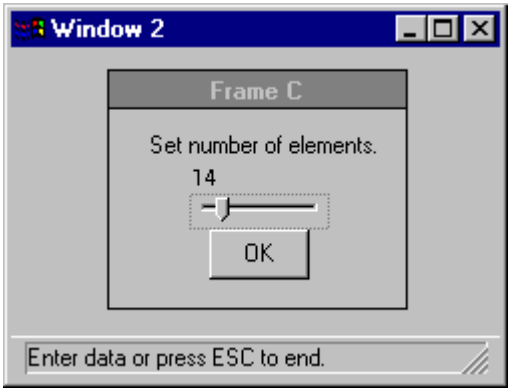
A window is a rectangular area on the screen that can contain frame widgets, parent dialog boxes, and parent other windows. It is surrounded by a standard border and affordances provided by your window system to manipulate the window's size, location, and appearance on the screen.

Progress automatically creates one default window for each session. You can create additional dynamic windows with the CREATE widget statement. Each additional window can be parented by the window system, creating siblings (the default) or by another window, creating child and parent window relationships. You create a parent and child relationship between two windows by setting the PARENT attribute of one (the child) to the widget handle of the other (the parent).

Windows in a parent and child relationship form a window family, which is a hierarchy of parent and child windows ultimately parented by the window system. The top parent window that is parented by the window system is the root window of the window family.

The following figure shows a window family consisting of a root window and its child window:





Attributes

(1 of 2)

ALWAYS-ON-TOP attribute[4]	BGCOLOR attribute[1]	COLUMN attribute
CONTEXT-HELP attribute[1,4]	CONTEXT-HELP-FILE attribute[1,4]	CONTROL-BOX attribute[4]
DCOLOR attribute[2]	DROP-TARGET attribute	DYNAMIC attribute[3]
FGCOLOR attribute[1]	FIRST-CHILD attribute[3]	FONT attribute[1]
FULL-HEIGHT-CHARS attribute[3]	FULL-HEIGHT-PIXELS attribute[3]	FULL-WIDTH-CHARS attribute[3]
FULL-WIDTH-PIXELS attribute[3]	HANDLE attribute[3]	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	HIDDEN attribute	HWND attribute[3,4]
ICON attribute	KEEP-FRAME-Z-ORDER attribute	LAST-CHILD attribute[3]
MAX-BUTTON attribute[4]	MAX-HEIGHT-CHARS attribute	MAX-HEIGHT-PIXELS attribute
MAX-WIDTH-CHARS attribute	MAX-WIDTH-PIXELS attribute	MENU-BAR attribute
MENU-KEY attribute	MENU-MOUSE attribute[1]	MESSAGE-AREA attribute[1,5]
MESSAGE-AREA-FONT attribute[1]	MIN-BUTTON attribute[4]	MIN-HEIGHT-CHARS attribute

(2 of 2)

MIN-HEIGHT-PIXELS attribute	MIN-WIDTH-CHARS attribute	MIN-WIDTH-PIXELS attribute
MOUSE-POINTER attribute	NAME attribute	NEXT-SIBLING attribute[3]
NUM-DROPPED-FILES attribute[3]	NUM-SELECTED-WIDGETS attribute[3]	PARENT attribute[1]
PFCOLOR attribute[2]	POPUP-MENU attribute	PREV-SIBLING attribute[3]
PRIVATE-DATA attribute	RESIZE attribute[1,5]	ROW attribute
SCREEN-LINES attribute	SCROLL-BARS attribute	SENSITIVE attribute
SHOW-IN-TASKBAR attribute[4,5]	SMALL-ICON attribute	SMALL-TITLE attribute[4,5]
STATUS-AREA attribute[1,5]	STATUS-AREA-FONT attribute[1]	THREE-D attribute[4]
TITLE attribute[5]	TOP-ONLY attribute[4]	TYPE attribute[3]
VIRTUAL-HEIGHT-CHARS attribute	VIRTUAL-HEIGHT-PIXELS attribute	VIRTUAL-WIDTH-CHARS attribute
VIRTUAL-WIDTH-PIXELS attribute	VISIBLE attribute	WIDTH-CHARS attribute
WIDTH-PIXELS attribute	WINDOW attribute[3]	WINDOW-STATE attribute
X attribute	Y attribute	

[1] Graphical interfaces only.

[2] Character interfaces only.

[3] Readable only.

[4] Windows only.

[5] Can be set only before the window is realized.

Methods

END-FILE-DROP( ) method	GET-DROPPED-FILE( ) method
GET-SELECTED-WIDGET( ) method	LOAD-ICON( ) method
LOAD-MOUSE-POINTER( ) method	LOAD-SMALL-ICON( ) method
MOVE-TO-BOTTOM( ) method	MOVE-TO-TOP( ) method

Events

Developer events	Mouse events
Universal key function events	DROP-FILE-NOTIFY
ENTRY	LEAVE
PARENT-WINDOW-CLOSE	WINDOW-CLOSE
WINDOW-MAXIMIZED	WINDOW-MINIMIZED
WINDOW-RESIZED	WINDOW-RESTORED

Notes

- Under the character user interface, you can have only one window (the default window).
- Certain manipulations of a parent window have a default effect on its child windows and their descendants. You cannot modify the following effects:
  - Iconifying a window (triggered by a WINDOW-MINIMIZED event or by setting the WINDOW-STATE attribute to WINDOW-MINIMIZED) causes any of its descendant windows that are not already iconified to be hidden. Any child windows that are already iconified remain iconified along with their parents. Restoring the parent window (triggered by a WINDOW-RESTORED event) causes all of its descendant windows to receive a WINDOW-RESTORED event, restoring them to their visual state prior to the parent window being minimized.
  - When a window receives a WINDOW-MINIMIZED event, all of its descendant windows receive WINDOW-MINIMIZED events. When a window receives a WINDOW-RESTORED event, all of its descendant windows receive WINDOW-RESTORED events.
  - Applying a WINDOW-CLOSE event to a window causes all of its descendant windows to receive a PARENT-WINDOW-CLOSE event. However, any ancestor windows remain unaffected.
- A WINDOW-MINIMIZED or WINDOW-RESTORED event applied to a child window has no effect on its parent or ancestor windows.

- Resizing or changing the position of a window has no affect on the size or position of any descendants or ancestors of that window.
- The following attributes have the Unknown value (?) until the window is realized:
  - FULL-HEIGHT-CHARS attribute
  - FULL-HEIGHT-PIXELS attribute
  - FULL-WIDTH-CHARS attribute
  - FULL-WIDTH-PIXELS attribute

See also DIALOG-BOX widget, the chapter on windows in *OpenEdge Development: Progress 4GL Handbook*.

Handle Reference

This chapter describes Progress object handles and lists the attributes and methods of each one. Object handles are essentially addresses that provide access to 4GL objects in memory. These objects include widgets, which encapsulate user interface capabilities, and other object types that provide access to a variety of OpenEdge session capabilities.

Progress makes object handles available in two ways:

- Directly, as system handle values. A system handle is a Progress keyword that evaluates to an object handle whose object type is implied by the keyword. For example, the CURRENT-WINDOW system handle is a handle to a window widget. To access the attributes and methods of a system handle, you can use the keyword directly, or you can assign the keyword value to a handle variable and use the variable to reference the attributes and methods:

```
DEFINE hHandle AS HANDLE NO-UNDO.
hHandle = THIS-PROCEDURE. /* Set a handle to a procedure object. */
DISPLAY THIS-PROCEDURE:GET-SIGNATURE(). /* Displays the same */
DISPLAY hHandle:GET-SIGNATURE(). /* signature as this statment. */
```

- Indirectly, as values output from certain Progress statements, including other handle attributes and methods. You can access the attributes and methods of any object by assigning its handle to a handle variable, which can hold handle values of any type. You can then use this variable to reference the attributes and methods of the object:

```
DEFINE hHandle AS HANDLE NO-UNDO.
CREATE SERVER hHandle. /* Create a handle to a server object. */
hHandle:CONNECT().     /* Execute a server object method. */
```

The following reference entries include both system handles and handle types for objects not necessarily referenced using system handles. Each system handle is listed by its keyword (all upper case, for example: SESSION system handle), and each handle type is listed by its type (upper/lower case, for example: Server object handle).

Each entry lists the attributes and methods supported by the handle or refers you to a more general entry with the same list. For example, the attributes and methods of the CURRENT-WINDOW system handle appear under the WINDOW widget entry. Widgets, in general, share a common set of user interface capabilities. For a list of the attributes and methods supported by each widget, see the "Widget Reference" section on page 1311.

For more information on how to access attributes and methods using all object handles, see the "Attributes and Methods Reference" section on page 1497, which includes a complete reference entry for each attribute and method.

Note: You may consider an object handle to be supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in the reference entry.

ACTIVE-WINDOW system handle

A handle to the last OpenEdge application window to receive an ENTRY event. You cannot set the ACTIVE-WINDOW handle, but you can read and write values for the attributes of the ACTIVE-WINDOW.

Syntax

```
ACTIVE-WINDOW [ :attribute ]
```

attribute

An attribute of the ACTIVE-WINDOW. The ACTIVE-WINDOW handle has all the attributes of a window widget.

Notes

- In a character interface, the ACTIVE-WINDOW, CURRENT-WINDOW, and DEFAULT-WINDOW handles return the handle of the static window for the current OpenEdge session.
- The initial value of the ACTIVE-WINDOW handle is the CURRENT-WINDOW handle.
- The ACTIVE-WINDOW handle monitors the active window in the OpenEdge session only. It does not monitor the active window for the window system. Accessing a non-Progress window does not affect the state of the ACTIVE-WINDOW handle.
- You can enable or disable the current window by changing ACTIVE-WINDOW:SENSITIVE.
- You can set the menu bar for the active window by assigning the handle of a menu bar to ACTIVE-WINDOW:MENUBAR.

See also

CURRENT-WINDOW system handle, DEFAULT-WINDOW system handle

Asynchronous request object handle

Maintains the status of an asynchronous request running on an AppServer or Web service.

Syntax

```
async-request-handle [ : attribute ]
```

async-request-handle

A handle variable that references an asynchronous request object. This object is instantiated when you execute an asynchronous remote procedure using the RUN statement specified with the ASYNCHRONOUS option. You can get the handle value by doing one of the following:

- Use the ASYNCHRONOUS SET option on the same RUN statement that instantiates the asynchronous request.
- Reference the LAST-ASYNC-REQUEST attribute on the server handle for the AppServer where the request is running. To ensure that you are referencing a specific request, you must reference the attribute after the associated RUN statement executes and before you instantiate another asynchronous request on the same AppServer connection.
- You can also locate the asynchronous request handle by walking the chain between the FIRST-ASYNC-REQUEST and LAST-ASYNC-REQUEST attributes of the associated server handle. Search on the PROCEDURE-NAME attribute of each request handle to identify the specific request.

attribute

An attribute of the asynchronous request handle.

Attributes

CANCELLED attribute	COMPLETE attribute	ERROR attribute
EVENT-PROCEDURE attribute	EVENT-PROCEDURE-CONTEXT attribute	NAME attribute
NEXT-SIBLING attribute	PERSISTENT-PROCEDURE attribute	PREV-SIBLING attribute
PRIVATE-DATA attribute	PROCEDURE-NAME attribute	QUIT attribute
SERVER attribute	STOP attribute	TYPE attribute

Events

PROCEDURE-COMPLETE

Notes

- When the AppServer completes and returns the results of the asynchronous request associated with this handle, the client application that executed the request receives the PROCEDURE-COMPLETE event. This event triggers execution of the associated event procedure (if specified) in the context of an I/O blocking statement, such as the WAIT-FOR statement, UPDATE statement, or a PROCESS EVENTS statement.
- You can access this handle anywhere in the client application that executes the associated request. However, it is especially useful for reference in the event procedure for the asynchronous request. In the associated event procedure, you can access this handle as the value of the SELF system handle.
- For more information on asynchronous requests, the PROCEDURE-COMPLETE event, and asynchronous request handles, see *OpenEdge Application Server: Developing AppServer Applications*.

See also

RUN statement, Server object handle, SAX-reader object handle, WAIT-FOR statement

AUDIT-CONTROL system handle

A handle to the audit control settings for managing application auditing context and events for the current OpenEdge session.

Syntax

```
AUDIT-CONTROL [ :attribute | :method ]
```

attribute

An attribute of the AUDIT-CONTROL system handle.

method

A method of the AUDIT-CONTROL system handle.

Attributes

APPL-CONTEXT-ID attribute	EVENT-GROUP-ID attribute

Methods

BEGIN-EVENT-GROUP( ) method	CLEAR-APPL-CONTEXT( ) method
END-EVENT-GROUP( ) method	LOG-AUDIT-EVENT( ) method
SET-APPL-CONTEXT( ) method	

See also

AUDIT-POLICY system handle, Client-principal object handle, SECURITY-POLICY system handle

AUDIT-POLICY system handle

A handle that lets you update current audit policy settings for processing audit events and securing audit data for an audit-enabled database.

Syntax

```
AUDIT-POLICY [ :method ]
```

method

A method of the AUDIT-POLICY system handle.

Methods

ENCRYPT-AUDIT-MAC-KEY( ) method	REFRESH-AUDIT-POLICY( ) method

For information about audit-enabling a database, or creating and activating an audit policy for a database, see *OpenEdge Getting Started: Core Business Services*.

See also

AUDIT-CONTROL system handle, AUDIT-ENABLED function, Client-principal object handle, SECURITY-POLICY system handle

Buffer object handle

A handle to a buffer object, corresponding to an underlying Progress buffer, which can be static or dynamic. An example of a static underlying buffer is one you define at compile time by using the DEFINE BUFFER statement, or by implicitly referencing a table in a 4GL construct such as "customer.cust-num". An example of a dynamic underlying buffer is one you create at run time with the CREATE BUFFER statement.

Syntax

```
buffer-handle [ :attribute | :method ]
```

buffer-handle

An item of type HANDLE representing a handle to a buffer object.

attribute

An attribute of the buffer object.

method

A method of the buffer object.

Attributes

(1 of 2)

ADM-DATA attribute	AFTER-BUFFER attribute	AFTER-ROWID attribute
AMBIGUOUS attribute	ATTACHED-PAIRLIST attribute	AUTO-DELETE attribute
AUTO-SYNCHRONIZE attribute	AVAILABLE attribute	BEFORE-BUFFER attribute
BEFORE-ROWID attribute	CAN-CREATE attribute	CAN-DELETE attribute
CAN-READ attribute	CAN-WRITE attribute	CRC-VALUE attribute
CURRENT-CHANGED attribute	DATASET attribute	DATA-SOURCE attribute
DATA-SOURCE-COMPLETE-MAP attribute	DATA-SOURCE-MODIFIED attribute	DBNAME attribute
DYNAMIC attribute	ERROR attribute	ERROR-STRING attribute
FILL-MODE attribute	HANDLE attribute	HAS-LOBS attribute
KEYS attribute	LOCKED attribute	NAME attribute

(2 of 2)

NAMESPACE-PREFIX attribute	NAMESPACE-URI attribute	NEW attribute
NEXT-SIBLING attribute	NUM-CHILD-RELATIONS attribute	NUM-FIELDS attribute
NUM-REFERENCES attribute	ORIGIN-ROWID attribute	PARENT-RELATION attribute
PRIMARY attribute	PRIVATE-DATA attribute	QUERY attribute
RECID attribute	RECORD-LENGTH attribute	REJECTED attribute
ROWID attribute	ROW-STATE attribute	TABLE attribute
TABLE-HANDLE attribute	TABLE-NUMBER attribute	TYPE attribute
UNIQUE-ID attribute		

Methods

(1 of 2)

ACCEPT-CHANGES( ) method	ACCEPT-ROW-CHANGES( ) method
APPLY-CALLBACK( ) method	ATTACH-DATA-SOURCE( ) method
BUFFER-COMPARE( ) method	BUFFER-COPY( ) method
BUFFER-CREATE( ) method	BUFFER-DELETE( ) method
BUFFER-FIELD( ) method	BUFFER-RELEASE( ) method
BUFFER-VALIDATE( ) method	DETACH-DATA-SOURCE( ) method
DISABLE-DUMP-TRIGGERS( ) method	DISABLE-LOAD-TRIGGERS( ) method
EMPTY-TEMP-TABLE( ) method	FILL( ) method
FIND-BY-ROWID( ) method	FIND-CURRENT( ) method
FIND-FIRST( ) method	FIND-LAST( ) method
FIND-UNIQUE( ) method	GET-CALLBACK-PROC-CONTEXT( ) method
GET-CALLBACK-PROC-NAME( ) method	GET-CHANGES( ) method
GET-CHILD-RELATION( ) method	INDEX-INFORMATION( ) method
MERGE-CHANGES( ) method	MERGE-ROW-CHANGES( ) method
RAW-TRANSFER( ) method	READ-XML( ) method
READ-XMLSCHEMA( ) method	REJECT-CHANGES( ) method

(2 of 2)

REJECT-ROW-CHANGES( ) method	SAVE-ROW-CHANGES( ) method
SET-CALLBACK( ) method	SET-CALLBACK-PROCEDURE( ) method
SYNCHRONIZE( ) method	WRITE-XML( ) method
WRITE-XMLSCHEMA( ) method	

Events

AFTER-FILL event	AFTER-ROW-FILL event
BEFORE-FILL event	BEFORE-ROW-FILL event
FIND-FAILED event	OFF-END event
ROW-CREATE event	ROW-DELETE event
SYNCHRONIZE event	

For information on these events, see the "ProDataSet events" section on page 2195.

Note For more information on the buffer object, see *OpenEdge Development: Progress 4GL Handbook*.

See also Buffer-field object handle, ProDataSet object handle, Query object handle, Temp-table object handle

Buffer-field object handle

A handle to a buffer-field object. Buffer-field objects let you examine and modify the fields of a record. They also let you examine the schema properties of the field.

Syntax

```
buffer-field-handle [ :attribute ]
```

buffer-field-handle

A handle to a buffer-field object.

attribute

An attribute of the buffer-field object.

Attributes

ADM-DATA attribute	AVAILABLE attribute	BUFFER-HANDLE attribute
BUFFER-NAME attribute	BUFFER-VALUE attribute	CAN-READ attribute
CAN-WRITE attribute	CASE-SENSITIVE attribute	COLUMN-LABEL attribute
DATA-TYPE attribute	DBNAME attribute	DECIMALS attribute
EXTENT attribute	FORMAT attribute	HANDLE attribute
HELP attribute	INITIAL attribute	KEY attribute
LABEL attribute	LITERAL-QUESTION attribute	MANDATORY attribute
NAME attribute	POSITION attribute	PRIVATE-DATA attribute
READ-ONLY attribute	STRING-VALUE attribute	TABLE attribute
TYPE attribute	UNIQUE-ID attribute	VALIDATE-EXPRESSION attribute
VALIDATE-MESSAGE attribute	WIDTH-CHARS attribute	XML-DATA-TYPE attribute
XML-NODE-TYPE attribute		

Note

For more information on the buffer-field object, see *OpenEdge Development: Progress 4GL Handbook*.

See also

Buffer object handle, Query object handle

CALL object handle

A handle to a CALL object, which lets you do the following dynamically:

- Invoke an external procedure, internal procedure, or user-defined function.
- Get or set an attribute.
- Run a method.

Syntax

```
call-object-handle [ :attribute | :method ]
```

call-object-handle

A handle to a CALL object.

attribute

An attribute of the CALL object.

method

A method of the CALL object. The methods let you set parameters, reset attributes to their default values, and invoke the CALL object.

Attributes

ASYNCHRONOUS attribute	ASYNC-REQUEST-HANDLE attribute	CALL-NAME attribute
CALL-TYPE attribute	EVENT-PROCEDURE attribute	EVENT-PROCEDURE-CONTEXT attribute
IN-HANDLE attribute	IS-PARAMETER-SET attribute	NUM-PARAMETERS attribute
PERSISTENT attribute	RETURN-VALUE attribute	RETURN-VALUE-DATA-TYPE attribute
SERVER attribute		

Methods

CLEAR( ) method	INVOKE( ) method
SET-PARAMETER( ) method	

Examples

The following fragment dynamically invokes an external procedure nonpersistently:

```
DEFINE VARIABLE hCall AS HANDLE.

CREATE CALL hCall.

/*invoke hello.p nonpersistently */
hCall:CALL-NAME = "hello.p".
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER( 1, "CHARACTER", "INPUT", "HELLO WORLD").
hCall:INVOKE.

/* clean up */
DELETE OBJECT hCall.
```

The following fragment dynamically invokes an external procedure persistently, then dynamically invokes one of its internal procedures:

```
DEFINE VARIABLE hCall AS HANDLE.

CREATE CALL hCall.

/*invoke persis.p persistently */
hCall:CALL-NAME = "persis.p".
hCall:PERSISTENT = true.
HCall:INVOKE.
   /* IN-HANDLE is automatically set
      to the handle of the persistent procedure
   */

/* invoke internal-persis-proc in persis.p */
HCall:CALL-NAME = "internal-persis-proc".
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER( 1, "INTEGER", "INPUT", 333).
hCall:INVOKE.

/* clean up */
DELETE PROCEDURE hCall:IN-HANDLE.
DELETE OBJECT hCall.
```

The following fragment uses a single CALL object handle multiple times:

```
DEFINE VARIABLE hCall AS HANDLE.

CREATE CALL hCall.

/*invoke hello.p nonpersistently */
hCall:CALL-NAME = "hello.p".
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER( 1, "CHARACTER", "INPUT", "HELLO WORLD").
hCall:INVOKE.

/* reset the CALL object handle */
hCall:CLEAR.

/*invoke persis.p persistently */
hCall:CALL-NAME = "persis.p".
hCall:PERSISTENT = true.
HCall:INVOKE.
   /* IN-HANDLE is automatically set
      to the handle of the persistent procedure
   */

/* invoke internal-persis-proc in persis.p
HCall:CALL-NAME = "internal-persis-proc".
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER( 1, "INTEGER", "INPUT", 333).
hCall:INVOKE.

/* clean up */
DELETE PROCEDURE hCall:IN-HANDLE.
DELETE OBJECT hCall.
```

The following fragment gets an attribute:

```
/* get title of frame */

hCall:IN-HANDLE = myframe_handle.
HCALL:CALL-TYPE = GET-ATTR-CALL-TYPE.
hCall:CALL-NAME = "TITLE".
hCall:INVOKE.
Mytitle = hCall:RETURN-VALUE.
```

The following fragment sets an attribute:

```
/* set SESSION:NUMERIC-FORMAT to "european" */
hCall:IN-HANDLE = "session".
HCALL:CALL-TYPE = SET-ATTR-CALL-TYPE.
hCall:CALL-NAME = "numeric-format".
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER( 1, "CHAR", "INPUT", "european").
hCall:INVOKE.
```

The following fragment drives the INVOKE() from a TEMP-TABLE:

```
/*suppose hRuntt is a temp-table that has one record with the following fields:
   parm_1
   parm_2
   ...
   parm_n
   run-name
   nparms
   datatypes, extent nparms
   iomodes, extent nparms
*/
DEFINE INPUT PARAMETER TABLE-HANDLE hRuntt.
DEFINE VARIABLE hDtypes  AS HANDLE.
DEFINE VARIABLE hIOmodes AS HANDLE.
DEFINE VARIABLE hCall    AS HANDLE.
DEFINE VARIABLE i        AS INTEGER.

hDtypes  = hRuntt:BUFFER-FIELD("datatypes").
hIOmodes = hRuntt:BUFFER-FIELD("iOmodes").

hRuntt:FIND-FIRST.

CREATE CALL hCall.
hCall:CALL-NAME = hRuntt:BUFFER-FIELD("run-name"):BUFFER-VALUE.
hCall:NUM-PARAMETERS = hRuntt:BUFFER-FIELD("nparms"):BUFFER-VALUE.

FOR i = 1 to hCall:NUM-PARAMETERS.
    hCall:SET-PARAMETER( i,
     hDtypes:BUFFER-VALUE(i),
     hIOmodes:BUFFER-VALUE(i),
     hRuntt:BUFFER-FIELD(i):BUFFER-VALUE).
END.

hCall:INVOKE.
DELETE OBJECT hCall.
/* if there are output parms,
   get the values from hRuntt:BUFFER-FIELD(i)
*/
```

Notes

- Invoking logic dynamically requires many more lines of code than does invoking it statically, which can make your program hard to read. For this reason, Progress Software Corporation recommends that you invoke logic dynamically only when absolutely necessary. Specifically, use the CALL object only when you cannot use the RUN statement, the DYNAMIC-FUNCTION() function, or *widget:attribute* or *widget:method* syntax, as in the following situations:

  – To invoke an internal or external procedure whose **calling sequence** (number of parameters and the data type of each) is unknown at compile time.

  Note: But if only the **name** of the procedure is unknown at compile time, use the RUN statement with the VALUE option—and avoid the CALL object altoghether.

  – To invoke a function whose calling sequence is unknown at compile time.

  Note: But if only the **name** of the function is unknown at compile time, use the DYNAMIC-FUNCTION() function—and avoid the CALL object altogether.

  – To reference a widget attribute or method whose name is unknown at compile time.

  If you already know the name of the attribute or procedure, you know its syntax, since the name implies certain syntax. And if you know the syntax, you know the calling sequence, since the syntax defines the calling sequence. And if you know the calling sequence, you can use *widget:attribute* or *widget:method* syntax—and avoid the CALL object altogether.

- To create a CALL object, use the following syntax:

```
CREATE object handle [ IN widget-pool ]
```

For example:

```
CREATE CALL hCall.
```

Note: Unlike most 4GL objects, the CALL object, by default, goes into the SESSION widget pool, not into the closest unnamed widget pool.

- To delete a CALL object, use the following syntax:

```
DELETE OBJECT handle
```

For example:

```
DELETE OBJECT CALL hCall.
```

Since the CALL object, by default, goes into the SESSION widget pool, not into the closest unnamed widget pool, to delete a CALL object created when the IN widget-pool option is not used, use the `DELETE object handle` syntax explicitly.

See also RUN statement

Client-principal object handle

A handle to a Client-principal object. Each Client-principal object instance contains information specific to one user login session. This login session may be used as an application or database user identity from an authentication domain registered in a trusted authentication domain registry. You create an instance of a Client-principal object at run time using the CREATE CLIENT-PRINCIPAL statement.

Note: You use a Client-principal object with the SET-CLIENT( ) method or SET-DB-CLIENT function to set the user identity for a OpenEdge session or OpenEdge database. You can have only one active Client-principal object set as the current user at any one point in time for a session or database connection.

Syntax

```
client-principal-handle [ :attribute | :method ]
```

client-principal-handle

A variable of type HANDLE that references a Client-principal object.

attribute

An attribute of the Client-principal object handle.

method

A method of the Client-principal object handle.

Attributes

AUDIT-EVENT-CONTEXT attribute	CLIENT-TTY attribute
CLIENT-WORKSTATION attribute	DOMAIN-DESCRIPTION attribute
DOMAIN-NAME attribute	DOMAIN-TYPE attribute
LOGIN-EXPIRATION-TIMESTAMP attribute	LOGIN-HOST attribute
LOGIN-STATE attribute	ROLES attribute
SEAL-TIMESTAMP attribute	SESSION-ID attribute
STATE-DETAIL attribute	USER-ID attribute

Methods

AUTHENTICATION-FAILED( ) method	EXPORT-PRINCIPAL( ) method
GET-PROPERTY( ) method	IMPORT-PRINCIPAL( ) method
LIST-PROPERTY-NAMES( ) method	LOGOUT( ) method
SEAL( ) method	SET-PROPERTY( ) method
VALIDATE-SEAL( ) method	

See also AUDIT-CONTROL system handle, AUDIT-POLICY system handle, CREATE CLIENT-PRINCIPAL statement, SECURITY-POLICY system handle, SET-CLIENT( ) method, SETUSERID function, SET-DB-CLIENT function,

CLIPBOARD system handle

A handle to the system clipboard widget. The CLIPBOARD handle allows you to implement interactions that allow the user to transfer data between Progress field-level widgets, or between Progress field-level widgets and the widgets of other applications running on the system. Progress can interpret the data read from or written to the system clipboard widget as a single item or as a group of multiple items. These data transfers are typically invoked as cut, copy, and paste operations.

Syntax

```
CLIPBOARD [ :attribute ]
```

attribute

An attribute of the clipboard widget.

Attributes

AVAILABLE-FORMATS attribute	ITEMS-PER-ROW attribute	MULTIPLE attribute
NUM-FORMATS attribute	TYPE attribute	VALUE attribute

Examples

The following code fragment implements cut, copy, and paste operations for the EM_Cut, EM_Copy, and EM_Paste items on the EditMenu menu. It uses the FOCUS handle to reference the widget that has the current input focus.

Note that the fragment tests the widget type of the FOCUS widget in two instances: once when EditMenu is opened during the MENU-DROP event to determine what clipboard operations are valid for the widget, and once again when a clipboard operation is chosen from the menu to determine how the operation is executed for the widget. During the MENU-DROP event, if a particular operation is valid for the FOCUS widget the menu item for that operation is enabled. Otherwise, it is disabled.

During the CHOOSE event for a n enabled menu item, the fragment executes the corresponding clipboard operation in a way that accounts for the unique features of the FOCUS widget. For example, the copy operation (EM_Copy) copies the selected text from an editor widget, copies the label text from a radio set item, and copies a composed true or false message for a toggle box. Your own implementation of these operations for the same widgets can be quite different.

For a complete description of this example, see the chapter on the system clipboard in *OpenEdge Development: Programming Interfaces*.

```
DEFINE VARIABLE Stat AS LOGICAL.
            .
            .
            .
ON MENU-DROP OF MENU EditMenu DO:
    IF FOCUS:TYPE = "EDITOR" THEN DO:
       MENU-ITEM EM_Cut:SENSITIVE IN MENU EditMenu =
          IF  LENGTH(FOCUS:SELECTION-TEXT) > 0
          THEN TRUE
          ELSE FALSE.
       MENU-ITEM Em_Copy:SENSITIVE IN MENU EditMenu =
          IF  LENGTH(FOCUS:SELECTION-TEXT) > 0
          THEN TRUE
          ELSE FALSE.
       MENU-ITEM EM_Paste:SENSITIVE IN MENU EditMenu =
          IF CLIPBOARD:NUM-FORMATS > 0
          THEN TRUE
          ELSE FALSE.
    END.
    ELSE IF FOCUS:TYPE = "RADIO-SET"      OR
            FOCUS:TYPE = "SELECTION-LIST" OR
            FOCUS:TYPE = "SLIDER"         OR
            FOCUS:TYPE = "TOGGLE-BOX" THEN DO:
       MENU-ITEM EM_Cut:SENSITIVE IN MENU EditMenu = FALSE.
       MENU-ITEM Em_Copy:SENSITIVE IN MENU EditMenu = TRUE.
       MENU-ITEM Em_Paste:SENSITIVE IN MENU EditMenu = FALSE.
    END.
    ELSE IF FOCUS:TYPE = "FILL-IN" THEN DO:
       MENU-ITEM EM_Cut:SENSITIVE IN MENU EditMenu =
          IF LENGTH(FOCUS:SCREEN-VALUE) > 0
          THEN TRUE
          ELSE FALSE.
       MENU-ITEM Em_Copy:SENSITIVE IN MENU EditMenu =
          IF LENGTH(FOCUS:SCREEN-VALUE) > 0
          THEN TRUE
          ELSE FALSE.
       MENU-ITEM EM_Paste:SENSITIVE IN MENU EDitMenu =
          IF CLIPBOARD:NUM-FORMATS > 0
          THEN TRUE
          ELSE FALSE.
    END.
    ELSE DO:
        MENU-ITEM EM_Cut:SENSITIVE IN MENU EditMenu = FALSE.
        MENU-ITEM EM_Copy:SENSITIVE IN MENU EditMenu = FALSE.
        MENU-ITEM EM_Paste:SENSITIVE IN MENU EditMenu = FALSE.
    END.
END. /* ON MENU-DROP IN EditMenu */
```

```
ON CHOOSE OF MENU-ITEM EM_Cut IN MENU EditMenu DO:
    IF FOCUS:TYPE = "EDITOR" THEN DO:
        IF FOCUS:SELECTION-START <> FOCUS:SELECTION-END THEN DO:
            CLIPBOARD:VALUE = FOCUS:SELECTION-TEXT.
            Stat = FOCUS:REPLACE-SELECTION-TEXT("").
        END.
        ELSE DO:
            CLIPBOARD:VALUE = FOCUS:SCREEN-VALUE.
            FOCUS:SCREEN-VALUE = "".
        END.
    END.
    ELSE DO: /* For FILL-IN */
        CLIPBOARD:VALUE = FOCUS:SCREEN-VALUE.
        FOCUS:SCREEN-VALUE = "".
    END.
END. /* ON CHOOSE OF MENU-ITEM EM_Cut */

ON CHOOSE OF MENU-ITEM EM_Copy IN MENU EditMenu DO:
    IF FOCUS:TYPE = "EDITOR" THEN
        IF FOCUS:SELECTION-START <> FOCUS:SELECTION-END THEN
            CLIPBOARD:VALUE = FOCUS:SELECTION-TEXT.
        ELSE
            CLIPBOARD:VALUE = FOCUS:SCREEN-VALUE.
    ELSE IF FOCUS:TYPE = "RADIO-SET" THEN
        CLIPBOARD:VALUE = ENTRY(LOOKUP(FOCUS:SCREEN-VALUE,
                                       FOCUS:RADIO-BUTTONS) - 1,
                                FOCUS:RADIO-BUTTONS).
    ELSE IF FOCUS:TYPE = "TOGGLE-BOX" THEN
        IF FOCUS:SCREEN-VALUE = "yes" THEN
            CLIPBOARD:VALUE = FOCUS:LABEL + " selected.".
        ELSE
            CLIPBOARD:VALUE = FOCUS:LABEL + " not selected.".
    ELSE /* For FILL-IN */
        CLIPBOARD:VALUE = FOCUS:SCREEN-VALUE.
END. /* ON CHOOSE OF MENU-ITEM EM_Copy */

ON CHOOSE OF MENU-ITEM EM_Paste IN MENU EditMenu DO:
    IF FOCUS:TYPE = "EDITOR" THEN DO:
        IF FOCUS:SELECTION-START <> FOCUS:SELECTION-END THEN
            Stat = FOCUS:REPLACE-SELECTION-TEXT(CLIPBOARD:VALUE).
        ELSE
            aResult = FOCUS:INSERT-STRING(CLIPBOARD:VALUE).
    END.
    ELSE /* For FILL-IN */
        FOCUS:SCREEN-VALUE = CLIPBOARD:VALUE.
END. /* ON CHOOSE OF MENU-ITEM EM_Paste */
            .
            .
            .
```

The following `r-clpmul.p` procedure demonstrates interaction with the clipboard using multiple items. The procedure copies out four rows of five numbers to the clipboard. It first displays the clipboard data as a single item, and then as a list of multiple items.

As a further demonstration of how the CLIPBOARD handle works with multiple items, try the following experiment:

1. Run the procedure, and at the pause, paste the result into an edit tool in your window system, such as Notepad in Windows.

2. You may have to select and copy text in the edit tool to activate the system clipboard before running the procedure.

3. Modify the text in the edit tool, leaving at least one tab or newline character, and copy it back to the clipboard from the edit tool.

4. Respond to the pause in the procedure to see how the modified clipboard data is displayed.

r-clpmul.p

```
DEFINE VARIABLE i AS INTEGER.
DEFINE VARIABLE ClipBuffer AS CHARACTER VIEW-AS EDITOR SIZE 60 BY 5.
DEFINE VARIABLE ClipItem AS CHARACTER.

/* Copy rows of integer items to the clipboard */
/* and display the clipboard value.            */
CLIPBOARD:MULTIPLE=TRUE.
CLIPBOARD:ITEMS-PER-ROW=5.
REPEAT i = 1 TO 20:
   CLIPBOARD:VALUE=STRING(i).
END.
CLIPBOARD:MULTIPLE=FALSE.
ClipBuffer = CLIPBOARD:VALUE.
ENABLE ClipBuffer WITH FRAME A.
DISPLAY SPACE(1) ClipBuffer LABEL "Clipboard Data" WITH FRAME A.
PAUSE.

/* Display each item of the clipboard value. */
CLIPBOARD:MULTIPLE=TRUE.
ClipItem="".
REPEAT WHILE ClipItem <> ?:
   ClipItem=CLIPBOARD:VALUE.
   IF ClipItem <> ? THEN
      DISPLAY SPACE(1) ClipItem
         FORMAT "x(16)" LABEL "Clipboard Item" WITH DOWN FRAME B.
END.
CLIPBOARD:MULTIPLE=FALSE.
```

Notes

- In character mode environments where there is no system clipboard, Progress supports CLIPBOARD handle operations within a single OpenEdge application. You can cut and paste among fields in one OpenEdge application, but not between one OpenEdge application and another OpenEdge or non-OpenEdge application.
- In graphical window systems, Progress supports CLIPBOARD handle operations using the system clipboard. This allows data transfers among OpenEdge and non-OpenEdge applications as well as within a single OpenEdge application.
- The AVAILABLE-FORMATS attribute returns a comma-separated string containing the names of the available formats for the data stored in the clipboard. Progress currently supports two formats:
  - **PRO_TEXT** — Specifies the standard text format on your system (CF_TEXT in Windows).
  - **PRO_MULTIPLE** — Specifies that the data in the clipboard contains tab or newline characters, and thus can be read as multiple items.
- The ITEMS-PER-ROW attribute specifies how Progress writes multiple items to the clipboard. Set the MULTIPLE attribute to TRUE before specifying ITEMS-PER-ROW. Then when you set ITEMS-PER-ROW to any integer value *n* greater than 1, Progress terminates every *n*th value you assign to the VALUE attribute with a newline character and terminates all other values with a tab character. This formats the output in the clipboard into newline-terminated rows of *n* items separated by tabs. The default value for the ITEMS-PER-ROW attribute is 1.

  During a MULTIPLE write, you can set and reset ITEMS-PER-ROW at any time until you set the MULTIPLE attribute to FALSE. When you set the MULTIPLE attribute to FALSE, Progress uses the current value of ITEMS-PER-ROW to format and flush the data to the clipboard, and resets the ITEMS-PER-ROW attribute to 1.

  The value of ITEMS-PER-ROW has no effect when reading data from the clipboard.

- The MULTIPLE attribute specifies whether Progress reads data from, and writes data to, the clipboard as a single item or as multiple items.

  When you set MULTIPLE to FALSE, Progress treats all data in the clipboard as a single item. Thus, any character string you assign to the VALUE attribute replaces all data in the clipboard, and whenever you read the VALUE attribute it returns all the data in the clipboard.

  When you set MULTIPLE to TRUE, Progress treats the data in the clipboard as multiple items separated by tab or newline characters.

  When you set the MULTIPLE attribute to TRUE and write values to the clipboard (assign values to the VALUE attribute), Progress stores the values in a buffer until you set MULTIPLE to FALSE. At this time Progress assigns the values to the clipboard separated from each other by tab or newline characters according to the value of the ITEMS-PER-ROW attribute. Note that the clipboard data itself does not change until you set MULTIPLE to FALSE. When you do set MULTIPLE to FALSE, all data previously in the clipboard is replaced by the items you have written.

  When you assign the MULTIPLE attribute to TRUE and read values from the clipboard (assign values from the VALUE attribute), each read returns the next item in the clipboard (starting with the first one). After all items have been read, the VALUE attribute returns the Unknown value (?). Setting the MULTIPLE attribute to FALSE and then to TRUE restarts the item pointer to read the first item of data in the clipboard.

  Until you (or another application) write data to the clipboard, changing the value of the MULTIPLE attribute itself has no effect on clipboard contents. It only affects the way you can access the clipboard for reading and writing.

  The default value for the MULTIPLE attribute is FALSE.

- The NUM-FORMATS attribute returns the number of formats available to read data from the clipboard. If no data is in the clipboard, the value is 0. If data is in the clipboard, the value is 1 (for PRO_TEXT) unless there are tab or newline characters in the data, in which case the value is 2 (for both PRO_TEXT and PRO_MULTIPLE).

- The VALUE attribute accesses the data in the clipboard. Reading the VALUE attribute has no effect on the clipboard contents. However, the exact value read or written depends on the setting of the MULTIPLE attribute.

  When the MULTIPLE attribute is FALSE, reading the VALUE attribute returns the current value in the clipboard as a single item. If there is no data in the clipboard, the VALUE attribute returns the Unknown value (?). Writing to the VALUE attribute immediately changes the current value in the clipboard to the value that is written.

  When the MULTIPLE attribute is TRUE, reading the VALUE attribute either references one of the multiple data items in the clipboard, or references the Unknown value (?) if all items have been read or there is no data in the clipboard. Writing to the VALUE attribute buffers each assignment and replaces the current data in the clipboard with the multiple values assigned when the MULTIPLE attribute is set to FALSE. See the previous description of the MULTIPLE attribute for more information.

  Note: Windows provides clipboard storage for a maximum of 64K of data.

  Assigning the Unknown value (?) to the VALUE attribute has no effect. To write a null item or clear the system clipboard when writing a single item, assign the null string ("") to the VALUE attribute.

- To cut or copy a Progress data item to the clipboard, set the CLIPBOARD:VALUE attribute to the value of the appropriate field or variable. A cut or copy operation replaces all data in the clipboard with the data from the specified Progress field or variable.

- To paste data from the clipboard to a Progress data item, assign the value of the CLIPBOARD:VALUE attribute to the appropriate the field or variable. If there is no data in the clipboard, a paste operation assigns the Unknown value (?) to the data item.

- To implement clipboard operations, use the FOCUS system handle, which identifies the Progress field-level widget that has the current input focus. Depending on the type of widget (for example, EDITOR or RADIO-ITEM) and its input state, you use one of several possible widget attributes as the source or destination for the data. For example, when working with selected text in an editor widget, use the SELECTION-TEXT attribute to cut or copy and the REPLACE-SELECTION-TEXT method to paste, but when working with the value of the entire editor field, use the SCREEN-VALUE attribute for all operations.

- Do not interrupt a 4GL clipboard operation with input blocking statements like UPDATE or WAIT-FOR. In general, make any 4GL clipboard cut, copy, or paste operation with the CLIPBOARD handle a one-step operation. Any interruption gives the user an opportunity to access and modify the clipboard from outside Progress, in the middle of the 4GL clipboard operation.
- Windows provides default clipboard operations through control keys, whether or not you implement them with the CLIPBOARD handle. These operations are available in editor and fill-in widgets, and are completely compatible with CLIPBOARD handle operations. They are single-item operations without any interaction with the MULTIPLE attribute. They also can occur in the middle of a 4GL clipboard operation, if it is interrupted. (See the previous bullet on interrupting 4GL clipboard operations.) The operations and control keys to activate them include:
  - Cut — **CTRL+X** and **SHIFT+DEL**.
  - Copy — **CTRL+C** and **CTRL+INS**.
  - Paste — **CTRL+V** and **SHIFT+INS**.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.
- For more information on implementing clipboard operations with the CLIPBOARD handle, see *OpenEdge Development: Programming Interfaces*.

See also FOCUS system handle

CODEBASE-LOCATOR system handle (Windows only; Graphical interfaces only)

A handle to the CODEBASE-LOCATOR object. A CODEBASE-LOCATOR object specifies the location and authentication information for a client application's codebase (that is, an application's files) stored on an AppServer or a web server. This object allows the WebClient to access application files for download. It also allows WebClient and the client application to share authentication information.

Note: Does not apply to SpeedScript programming.

Syntax

```
CODEBASE-LOCATOR [ :attribute ]
```

attribute

Specifies an attribute of the CODEBASE-LOCATOR handle.

Attributes

APPSERVER-INFO attribute	APPSERVER-PASSWORD attribute	APPSERVER-USERID attribute
END-USER-PROMPT attribute	KEEP-CONNECTION-OPEN attribute	KEEP-SECURITY-CACHE attribute
LOCATOR-TYPE attribute	NEEDS-APPSERVER-PROMPT attribute	NEEDS-PROMPT attribute
PERSISTENT-CACHE-DISABLED attribute	SERVER attribute	TYPE attribute
URL attribute	URL-PASSWORD attribute	URL-USERID attribute

Notes

- The CODEBASE-LOCATOR handle applies only to WebClient.
- The following attributes represent the security cache: APPSERVER-INFO, APPSERVER-PASSWORD, APPSERVER-USERID, URL-PASSWORD, URL-USERID, and KEEP-SECURITY-CACHE. These attributes are readable and writeable.
- WebClient sets the following read-only attributes based on values stored in the application configuration (`.ProwcApp`) file: END-USER-PROMPT, KEEP-CONNECTION-OPEN, LOCATOR-TYPE, NEEDS-APPSERVER-PROMPT, NEEDS-PROMPT, PERSISTENT-CACHE-DISABLED, and URL.
- Valid URL protocols depend on the LOCATOR-TYPE. If LOCATOR-TYPE is "`AppServer`", valid URL protocols include: HTTP, HTTPS, and AppServer. If LOCATOR-TYPE is "`InternetServer`", valid URL protocols include: HTTP, HTTPS, and FILE.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

See also CONNECT( ) method

COLOR-TABLE system handle (Windows only; Graphical interfaces only)

A handle to the current color table.

Note: Does not apply to SpeedScript programming.

Syntax

```
COLOR-TABLE [ :attribute | :method ]
```

attribute

An attribute of the COLOR-TABLE handle.

method

A method of the COLOR-TABLE handle.

Attributes

NUM-ENTRIES attribute	TYPE attribute

Methods

GET-BLUE-VALUE( ) method	GET-DYNAMIC( ) method
GET-GREEN-VALUE( ) method	GET-RED-VALUE( ) method
GET-RGB-VALUE( ) method	SET-BLUE-VALUE( ) method
SET-DYNAMIC( ) method	SET-GREEN-VALUE( ) method
SET-RED-VALUE( ) method	SET-RGB-VALUE( ) method

Example

This procedure sets the number of entries in the color table, makes color i + 1 dynamic, then sets the red, green, and blue values for this entry:

r-colhan.p

```
DEFINE VARIABLE red AS INTEGER INIT 0.
DEFINE VARIABLE blue AS INTEGER INIT 127.
DEFINE VARIABLE green AS INTEGER INIT 127.
DEFINE VARIABLE i AS INTEGER.

i = COLOR-TABLE:NUM-ENTRIES.
COLOR-TABLE:NUM-ENTRIES = i + 1.
COLOR-TABLE:SET-DYNAMIC(i, yes).
COLOR-TABLE:SET-RED-VALUE(i, red).
COLOR-TABLE:SET-GREEN-VALUE(i, green).
COLOR-TABLE:SET-BLUE-VALUE(i, blue).

DISPLAY COLOR-TABLE:GET-RED-VALUE (i).
DISPLAY COLOR-TABLE:GET-GREEN-VALUE (i).
DISPLAY COLOR-TABLE:GET-BLUE-VALUE (i).
```

Note: In this procedure, you can replace the SET-RED-VALUE( ), SET-GREEN-VALUE( ), and SET-BLUE-VALUE( ) methods with the SET-RGB-VALUE( ) method as follows:
`COLOR-TABLE:SET-RGB-VALUE(i, RGB-VALUE(red, green, blue)).`

Notes

- The current color table is the color table in the startup environment or the environment most recently specified in a USE statement.
- To determine the number of entries in the color table, access the NUM-ENTRIES attribute. For character interfaces, the value of this attribute is zero.
- To change the number of entries in the color table, modify the NUM-ENTRIES attribute.
- To let users modify color table entries at run time, display the System Color dialog box by coding the SYSTEM-DIALOG COLOR statement.
- To specify a red, green, or blue value for a dynamic color, supply an INTEGER expression that returns a value between 0 and 255 inclusive.

- To save a color definition from the color table to the current environment, use the PUT-KEY-VALUE statement. To retrieve a color definition from the current environment, use the GET-KEY-VALUE statement.
- The value of COLOR-TABLE:TYPE is “PSEUDO-WIDGET.”
- The SET-RGB-VALUE() and GET-RGB-VALUE() methods can be used as an alternative to specifying each individual red, green, and blue color value with the individual SET-RED-VALUE(), SET-GREEN-VALUE(), SET-BLUE-VALUE() methods, and GET-RED-VALUE(), GET-GREEN-VALUE(), and GET-BLUE-VALUE() methods, respectively.
- The SET-RGB-VALUE() and GET-RGB-VALUE() methods to set or retrieve colors are primarily used for Active X controls.
- The index is zero based. For example, the statement `COLOR-TABLE:GET-BLUE-VALUE(2)` returns the color of the 3rd entry.

See also GET-KEY-VALUE statement, PUT-KEY-VALUE statement, SYSTEM-DIALOG COLOR statement, USE statement

COM-SELF system handle (Windows only)

A component handle to the ActiveX object (ActiveX control or ActiveX automation object) that generated the event being handled by the currently executing ActiveX event procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
COM-SELF [ :OCX-property-reference | :OCX-method-reference ]
```

OCX-property-reference | *OCX-method-reference*

A reference to a valid property or method associated with the ActiveX control.

Example

The following code fragment displays the name and position of the ActiveX control that generates a Click event:

```
PROCEDURE ANYWHERE.Click:
  MESSAGE "Clicked control" COM-SELF:Name "at X-position" COM-SELF:Left
          "and Y-position" COM-SELF:Top VIEW-AS ALERT-BOX.
END PROCEDURE.
```

Notes

- Unlike 4GL widget handles that have the WIDGET-HANDLE data type, the component handle returned by COM-SELF has the COM-HANDLE data type.
- You can reference the COM-SELF handle only within an ActiveX control (OCX) event procedure.
- The syntax for referencing ActiveX control properties and methods extends the syntax for referencing widget attributes and methods. For more information, see the "Attributes and Methods Reference" section on page 1497.

See also

PROCEDURE statement, SAX-reader object handle

COMPILER system handle

A handle to information on a preceding COMPILE statement.

Syntax

```
COMPILER [ :attribute ]
```

attribute

Specifies an attribute of the COMPILER system handle.

Attributes

CLASS-TYPE attribute	ERROR attribute	ERROR-COLUMN attribute
ERROR-ROW attribute	FILE-NAME attribute	FILE-OFFSET attribute
MULTI-COMPILE attribute	STOPPED attribute	TYPE attribute
WARNING attribute		

Example

The input for example procedure is as input a comma-separated list of source files. It compiles each of these procedures. If a compilation error occurs, an appropriate message is written to the `compile.msgs` file.

r-cmpchk.p

```
/* Compile a series of source files passed in a comma separated list. */

DEFINE INPUT PARAMETER sources AS CHARACTER.
DEFINE VARIABLE entry-num AS INTEGER.

/* If the output file already exists, delete it.
   (If this results in an error, ignore the error.) */
OS-DELETE "compile.msgs".

DO entry-num = 1 TO NUM-ENTRIES(sources):
   COMPILE VALUE(ENTRY(entry-num, sources)) SAVE.
   IF COMPILER:ERROR
   THEN DO:
    OUTPUT TO "compile.msgs" APPEND.
    MESSAGE "Compilation error in" COMPILER:FILENAME "at line"
       COMPILER:ERROR-ROW "column" COMPILER:ERROR-COL.
    OUTPUT CLOSE.
   END.
END.
```

Notes

- If a compilation is successful, the COMPILER:ERROR attribute is set to FALSE.
- After a COMPILE statement, check the COMPILER:ERROR and COMPILER:WARNING attributes to determine whether the compilation was successful. If the value of ERROR is TRUE, you can use the FILE-NAME to determine in which source file the error occurred. You can use either the ERROR-ROW and ERROR-COLUMN attributes or the FILE-OFFSET attribute to determine where in the source file an error occurred. You can use this information to compose a message to display or write to a log file. To find the specific error and warning messages, check the ERROR-STATUS handle.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

See also COMPILE statement

CURRENT-WINDOW system handle

A handle to the default window for the current OpenEdge session. This window is the default parent for all frames, dialog boxes, alert boxes, and messages. Set or examine the attributes of the CURRENT-WINDOW handle to modify or get information on the current default window.

Syntax

```
CURRENT-WINDOW [ :attribute ]
```

attribute

An attribute of the CURRENT-WINDOW.

Attributes

The CURRENT-WINDOW handle has all the attributes of a window widget.

Notes

- The default value of the CURRENT-WINDOW handle is the static session window referenced by the DEFAULT-WINDOW handle. You can change the default window for the current session by assigning the handle of a window to CURRENT WINDOW.
- The IN WINDOW phrase allows you to explicitly assign a window as a parent for a frame, dialog box, alert box, or message.
- In a character interface, the ACTIVE-WINDOW, CURRENT-WINDOW, and DEFAULT-WINDOW handles return the handle of the static window for the current OpenEdge session.
- The CURRENT-WINDOW attribute of a procedure allows you to specify a default window for the procedure block. The CURRENT-WINDOW attribute of a procedure overrides the CURRENT-WINDOW handle for the procedure block.
- You can enable or disable the current window by changing CURRENT-WINDOW:SENSITIVE.
- You can set the menu bar for the current window by assigning the handle of a menu bar to CURRENT-WINDOW:MENUBAR.
- You can make the current window visible or invisible by changing the value of CURRENT-WINDOW:VISIBLE.

See also

ACTIVE-WINDOW system handle, DEFAULT-WINDOW system handle

Data-relation object handle

A handle to a data-relation object. A data-relation object defines one relation between a pair of parent and child buffers in a ProDataSet object that have a one-to-one or one-to-many parent-child relationship. A data-relation object identifies the parent and child buffers, and the fields in each buffer that define the primary and foreign key fields of the relation.

A data-relation object can be static or dynamic. A static data-relation object is one you define at compile time as part of the DEFINE DATASET statement using the data-relation option. A dynamic data-relation object is one you create at run time for a dynamic ProDataSet object using the ADD-RELATION( ) method. Use this handle to access the data-relation object's attributes.

Note: Does not apply to SpeedScript programming.

Syntax

```
data-relation-handle [ :attribute ]
```

data-relation-handle

A handle variable that references a data-relation object.

attribute

An attribute of the data-relation object.

Attributes

ACTIVE attribute	ADM-DATA attribute	CHILD-BUFFER attribute
HANDLE attribute	INSTANTIATING-PROCEDURE attribute	NAME attribute
NESTED attribute	PARENT-BUFFER attribute	PRIVATE-DATA attribute
QUERY attribute	RELATION-FIELDS attribute	REPOSITION attribute
TYPE attribute	WHERE-STRING attribute	

Notes

- You cannot define multiple data-relation objects for the same pair of parent and child buffers.
- You cannot delete a data-relation object. When the ProDataSet object is deleted or cleared, the data-relation objects are automatically deleted.

See also ADD-RELATION( ) method, DEFINE DATASET statement

Data-source object handle

A handle to a data-source object. A data-source object supports the automatic filling of a ProDataSet object member table, as well as applying updates back to one or more database tables. There is a distinct data-source object for each member buffer, which allows a single ProDataSet object and a single FILL operation on that object to combine data from multiple databases. A data-source object is defined independently of any ProDataSet object.

A data-source object can be static or dynamic. A static data-source object is one you define at compile time with the DEFINE DATA-SOURCE statement. A dynamic data-source object is one you create at run time with the CREATE DATA-SOURCE statement. Use this handle to access the data-source object's attributes and methods.

Note: Does not apply to SpeedScript programming.

Syntax

```
data-source-handle [ :attribute | :method ]
```

data-source-handle

A handle variable that references a data-source object.

attribute

An attribute of the data-source object.

method

A method of the data-source object.

Attributes

(1 of 2)

ADM-DATA attribute	FILL-WHERE-STRING attribute	HANDLE attribute
INSTANTIATING-PROCEDURE attribute	KEYS attribute	MERGE-BY-FIELD attribute
NAME attribute	NEXT-ROWID attribute	NEXT-SIBLING attribute
NUM-SOURCE-BUFFERS attribute	PREFER-DATASET attribute	PRIVATE-DATA attribute

(2 of 2)

QUERY attribute	RESTART-ROWID attribute	SAVE-WHERE-STRING attribute
TYPE attribute		

Methods

ADD-SOURCE-BUFFER( ) method	GET-DATASET-BUFFER( ) method
GET-SOURCE-BUFFER( ) method	

Notes

- To associate a query with a dynamic data-source object, use the QUERY attribute. To disassociate the query and data-source object, set the QUERY attribute to the Unknown value (?).
- To override the WHERE clause in the query, use the FILL-WHERE-STRING attribute.

See also CREATE DATA-SOURCE statement, DEFINE DATA-SOURCE statement

DEBUGGER system handle

A handle that lets 4GL procedures initialize and control the Application Debugger.

To use the DEBUGGER handle, you must have the Application Debugger installed in your OpenEdge environment.

Note: Does not apply to SpeedScript programming.

Syntax

```
DEBUGGER [ :attribute | :method ]
```

attribute

Specifies an attribute of the DEBUGGER handle.

method

Specifies a method of the DEBUGGER handle.

Attributes

TYPE attribute	VISIBLE attribute

Methods

CANCEL-BREAK( ) method	CLEAR( ) method
DEBUG( ) method	DISPLAY-MESSAGE( ) method
INITIATE( ) method	SET-BREAK( ) method

Example

The following example displays orders for each customer in the sports database using two procedure files. The `r-cusbug.p` file initializes the Debugger and sets a breakpoint at line 6 of the `r-ordbug.p` file. Thus, each time `r-ordbug.p` displays an order, the Debugger takes control before it displays the order lines. Just before completing execution, `r-cusbug.p` clears the debugging session before returning.

r-cusbug.p

```
DEFINE NEW SHARED BUFFER CustBuf FOR customer.

DEFINE VARIABLE debug AS LOGICAL.
debug = DEBUGGER:INITIATE().
debug = DEBUGGER:SET-BREAK("r-ordbug.p",6).

FOR EACH CustBuf:
    IF CAN-FIND(order OF CustBuf) THEN
        RUN r-ordbug.p.
END. /* FOR EACH CustBuf */

debug = DEBUGGER:CLEAR().
```

r-ordbug.p

```
DEFINE SHARED BUFFER CustBuf FOR customer.

FOR EACH order OF CustBuf:
    DISPLAY CustBuf.Name CustBuf.Cust-num CustBuf.City CustBuf.State
            CustBuf.Postal-code order.Order-num.
    FOR EACH order-line OF order, item OF order-line:
        DISPLAY item.Item-name item.Item-num order-line.Qty.
    END. /* FOR EACH order-line */
END. /* FOR EACH order */
```

Notes

- You must initialize the Debugger using either the DEBUG( ) or INITIATE( ) method before using any of the remaining methods in a procedure.

  The DEBUG( ) and INITIATE( ) methods provide separate means to invoke the Debugger, and do not depend on each other to start a debugging session. The DEBUG( ) method initializes and gives control to the Debugger whether or not the INITIATE( ) method has been executed.

- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

- The VISIBLE attribute specifies whether the Debugger window is visible on the screen. When set to FALSE, if the Debugger window is currently visible, it is removed from the screen. When set to TRUE, if the Debugger window is currently invisible, it is displayed. Note that making the Debugger window visible does not, in itself, give control to the Debugger.

  Note: The 4GL code that initiates the Debugger and displays it on the screen is responsible for removing the Debugger from the screen when it is no longer needed by setting the VISIBLE attribute to FALSE.

- After invoking the INITIATE method, execution continues in the procedure until it encounters a breakpoint or a statement invoking the DEBUG method. If the procedure encounters a breakpoint, the Debugger takes control running in application mode (with control over the invoking application). If the procedure invokes the DEBUG method, the Debugger takes control running in stand-alone mode (with control only over applications started from the Debugger).

- References to line numbers in internal procedures must be relative to the debug listing in which they are contained.

- When you set or cancel a breakpoint, you must distinguish between a line number value less than 1 and a value of 1 or greater. Any value for *line-number* less than 1 (for example, 0 or -1) specifies the first executable line of the **main procedure** in the file specified by *procedure*. However, a positive value for *line-number* specifies the first executable line on or after *line-number* in the **file** specified by *procedure*. For example, suppose *procedure* specifies a file like this:

```
1       DEFINE VARIABLE lStart AS LOGICAL INITIAL TRUE.
2       PROCEDURE ShowStart:
3           IF lStart THEN MESSAGE "Procedure is starting ...".
4       END.
5
6       MESSAGE "Hello World!".
7       RUN ShowStart.
8       lStart = FALSE.
                            .
                            .
```

If you specify a breakpoint at line 0, -1, or any negative value, the breakpoint actually occurs at line 6, the first line that executes in the main procedure. If you specify a breakpoint at line 1 or 2, the breakpoint occurs at line 3, the first executable line in the file, which happens to be the first executable line of an internal procedure.

This distinction also affects procedures containing the Trigger phrase used to define triggers in widget definitions. For example, suppose *procedure* specifies this file.

```
1       DEFINE BUTTON bOK LABEL "OK"
2           TRIGGERS:
3               ON CHOOSE
4                   MESSAGE "OK Pressed!".
5           END TRIGGERS.
6       MESSAGE "Hello World!".
                    .
                    .
```

Again, if you specify a breakpoint at line -1, the breakpoint occurs on line 6, but if you specify the breakpoint at line 1, it actually occurs at line 4, which is the first executable line of a trigger block.

Note: You cannot set a watchpoint programmatically using the DEBUGGER system handle. A watchpoint is a form of breakpoint which tells the Debugger to interrupt program execution when the value of a variable, buffer field, or attribute reference changes.

For more information on the Debugger, its features and functions, and its modes of execution, see *OpenEdge Development: Debugging and Troubleshooting*.

See also LOG-MANAGER system handle

DEFAULT-WINDOW system handle

A handle to the static window of the current OpenEdge session. Every OpenEdge session has one static window. Use the DEFAULT-WINDOW handle to set or examine the attributes of the unnamed session window.

Syntax

```
DEFAULT-WINDOW [ :attribute ]
```

attribute

An attribute of the DEFAULT-WINDOW.

Attributes

The DEFAULT-WINDOW handle has all the attributes of a window widget.

Notes

- In a character interface, the ACTIVE-WINDOW, CURRENT-WINDOW, and DEFAULT-WINDOW handles return the handle of the static window for the current OpenEdge session.
- You can make the default window the current window by assigning DEFAULT-WINDOW to CURRENT-WINDOW.
- You can enable or disable the default window by changing the value of DEFAULT-WINDOW:SENSITIVE (the SENSITIVE attribute).
- You can set the menu bar for the default window by assigning the handle of a menu bar to DEFAULT-WINDOW:MENUBAR (the MENUBAR attribute).
- You can make the default window visible or invisible by changing the value of DEFAULT-WINDOW:VISIBLE (the VISIBLE attribute).

See also

ACTIVE-WINDOW system handle, CURRENT-WINDOW system handle

ERROR-STATUS system handle

A handle to error information on the last statement executed with the NO-ERROR option.

Syntax

```
ERROR-STATUS [ :attribute | :method ]
```

attribute

Specifies an attribute of the ERROR-STATUS handle.

method

Specifies a method of the ERROR-STATUS handle.

Attributes

ERROR attribute	ERROR-OBJECT-DETAIL attribute	NUM-MESSAGES attribute
TYPE attribute		

Methods

GET-MESSAGE( ) method	GET-NUMBER( ) method

Examples

The following example uses the NO-ERROR and the ERROR-STATUS handle extensively to demonstrate when ERROR-STATUS attributes are reset:

r-errst1.p

```
CONNECT "db-xyz" NO-ERROR.
RUN chk-connect NO-ERROR.
IF ERROR-STATUS:ERROR
THEN MESSAGE "Run statement failed.".

PROCEDURE chk-connect.
DEFINE VARIABLE connect-ok AS LOGICAL INITIAL TRUE NO-UNDO.

   IF ERROR-STATUS:ERROR
   THEN DO:
       MESSAGE "Connect failed.".
       connect-ok = FALSE NO-ERROR.
       IF ERROR-STATUS:ERROR
       THEN MESSAGE "Assignment failed.".
   END.

   IF connect-ok
       THEN RETURN "OK".
   ELSE RETURN "FAILED".
END PROCEDURE.
```

Within the internal procedure, chk-connect, the first reference to ERROR-STATUS:ERROR returns status on the CONNECT statement from the main procedure. The second reference returns status on the assignment statement. The reference to ERROR-STATUS:ERROR in the main procedure returns status on the RUN statement. Note that the ERROR-STATUS attributes are set only after the statement with NO-ERROR completes. Therefore the references in the internal procedure are not affected by the RUN statement itself.

The following procedure accepts a character string value and lets you convert it to one of several data types. The internal convert procedure attempts the conversion. If the conversion is successful, it displays the converted value. If the conversion is unsuccessful, the ERROR-STATUS handle holds error information. After running convert, the CHOOSE trigger checks ERROR-STATUS:ERROR and ERROR-STATUS:NUM-MESSAGES to determine if error information is available. If it is, it lets you view this information.

r-errsts.p *(1 of 2)*

```
DEFINE VARIABLE txt AS CHARACTER FORMAT "X(20)" NO-UNDO.
DEFINE VARIABLE i   AS INTEGER NO-UNDO.

DEFINE BUTTON b_int  LABEL "Integer".
DEFINE BUTTON b_date LABEL "Date".
DEFINE BUTTON b_dec  LABEL "Decimal".
DEFINE BUTTON b_log  LABEL "Logical".
DEFINE BUTTON b_quit LABEL "Quit" AUTO-ENDKEY.

DEFINE FRAME butt-frame
  b_int b_date b_dec b_log b_quit
  WITH CENTERED ROW SCREEN-LINES - 2.DEFINE FRAME get-info
  txt LABEL "Enter Data To Convert"
  WITH ROW 2 CENTERED SIDE-LABELS TITLE "Data Conversion - Error Check".

ON CHOOSE OF b_int, b_date, b_dec, b_log IN FRAME butt-frame
DO:
  IF txt:MODIFIED IN FRAME get-info THEN
  DO:
    ASSIGN txt.
    RUN convert(txt).
    IF ERROR-STATUS:ERROR AND ERROR-STATUS:NUM-MESSAGES > 0 THEN
    DO:
      MESSAGE ERROR-STATUS:NUM-MESSAGES
       " errors occurred during conversion." SKIP
       "Do you want to view them?"
       VIEW-AS ALERT-BOX QUESTION BUTTONS YES-NO
       UPDATE view-errs AS LOGICAL.

      IF view-errs THEN
      DO i = 1 TO ERROR-STATUS:NUM-MESSAGES:
        MESSAGE ERROR-STATUS:GET-NUMBER(i)
                ERROR-STATUS:GET-MESSAGE(i).
      END.
    END.
  END. /* IF txt:MODIFIED... */
  ELSE
    MESSAGE "Please enter data to be convert and then choose the type"
            "of conversion to perform."
      VIEW-AS ALERT-BOX MESSAGE BUTTONS OK.
END.

ENABLE ALL WITH FRAME butt-frame.
ENABLE txt WITH FRAME get-info.
WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame FOCUS txt IN FRAME get-info.
```

r-errsts.p *(2 of 2)*

```
/***** Internal Procedure "convert" *****/
PROCEDURE convert:
  DEFINE INPUT PARAMETER x AS CHARACTER NO-UNDO.
  DEFINE VARIABLE i  AS INTEGER   NO-UNDO.
  DEFINE VARIABLE d  AS DECIMAL   NO-UNDO.
  DEFINE VARIABLE dd AS DATE      NO-UNDO.
  DEFINE VARIABLE l  AS LOGICAL   NO-UNDO.

  message self:label.

  CASE SELF:LABEL:
    WHEN "Integer" THEN DO:
      ASSIGN i = INT(x) NO-ERROR.
      MESSAGE "Converted value:" i.
    END.
    WHEN "Date" THEN DO:
      ASSIGN dd = DATE(INT(SUBSTR(x,1,2)),
                       INT(SUBSTR(x,4,2)),
                       INT(SUBSTR(x,7)) ) NO-ERROR.
      MESSAGE "Converted value:" dd.
    END.
    WHEN "Decimal" THEN DO:
      ASSIGN d = DEC(x) NO-ERROR.
      MESSAGE "Converted value:" d.
    END.
    WHEN "Logical" THEN DO:
      ASSIGN l = x = "yes" OR x = "true" NO-ERROR.
      MESSAGE "Converted value:" l.
    END.
  END.
END.
```

Notes

- The ERROR attribute indicates whether the ERROR condition was raised during the execution of the last statement that contained the NO-ERROR option. Some errors may occur without raising the ERROR condition. For example, compiler errors do not raise the ERROR condition.
- The NUM-MESSAGES attribute indicates the total number of errors that occurred during that statement.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

- The ERROR-OBJECT-DETAIL attribute identifies a SOAP-fault object that contains SOAP fault message detail.

  If a Web service operation generates a SOAP fault message, Progress generates the following error:

  ```
  Web service %s<operation> failed.  SOAP faultstring is %s (nnnn)
  ```

  The complete SOAP fault error message is returned to the 4GL as part of the ERROR-STATUS system handle.

  If the 4GL application invokes the Web service operation with the NO-ERROR option on the RUN statement, any errors that occur as a result of the operation are suppressed. In this case, the application can access the SOAP fault message detail using the SOAP-fault and SOAP-fault-detail object handles. Otherwise, Progress displays the error message to the end user.

- The GET-MESSAGE method and the GET-NUMBER method let you access the error numbers and messages for all errors that occurred during the execution of the last statement with the NO-ERROR option.

- Usually, the NO-ERROR option on a statement suppresses the display of error messages. However, if a STOP condition occurs, the error message is written to the windows. These messages are also available through the ERROR-STATUS attributes. For example, the STOP condition is raised when a procedure to be run is not found. Two specific instances of this are:

  – If you use NO-ERROR on a RUN statement and the procedure is not found or cannot compile.

  – If you execute a data handling statement, such as DELETE with the NO-ERROR option and the corresponding trigger procedure is not found or cannot compile.

See also SOAP-fault object handle, SOAP-fault-detail object handle

FILE-INFO system handle

A handle to an operating system file.

Syntax

```
FILE-INFO [ :attribute ]
```

attribute

Specifies an attribute of the FILE-INFO handle.

Attributes

FILE-CREATE-DATE attribute	FILE-CREATE-TIME attribute	FILE-MOD-DATE attribute
FILE-MOD-TIME attribute	FILE-NAME attribute	FILE-SIZE attribute
FILE-TYPE attribute	FULL-PATHNAME attribute	PATHNAME attribute
TYPE attribute		

Example

After you set the value of the FILE-NAME attribute, you can read the values of the other attributes. For example:

r-osfile.p

```
DEFINE VARIABLE os-file  AS CHARACTER FORMAT "x(60)" LABEL "File".

REPEAT:
  SET os-file WITH FRAME osfile-info.
  FILE-INFO:FILE-NAME = os-file.
  DISPLAY FILE-INFO:FULL-PATHNAME FORMAT "x(60)" LABEL "Full Path"
    FILE-INFO:PATHNAME FORMAT "x(60)" LABEL "Path"
    FILE-INFO:FILE-TYPE LABEL "Type"
    WITH FRAME osfile-info SIDE-LABELS TITLE "OS File Info".
END.
```

Notes

- You cannot use the FILE-INFO handle to by-pass operating system security. You must have read access to the file and the directory that contains it to obtain information through FILE-INFO.
- These attributes return the Unknown value (?) until they are set, and also if the specified file cannot be found or you do not have permission to access the file.
- If you set the FILE-NAME attribute to a relative pathname, the FILE-INFO handle searches the current PROPATH to locate the file.
- The FILE-TYPE attribute returns a string containing exactly one of the following file type characters:
  - **D** — If the file is a directory.
  - **F** — If the file is a standard file or FIFO pipe (UNIX systems).
  - **M** — If the file is a member of a Progress procedure library.
  - **S** — If the file is a special device (UNIX systems).
  - **X** — If the file type is unknown. (Contact your Progress Technical Support representative if you receive this value.)

  The attribute string can contain any of the following file type characters:
  - **H** — If the file is hidden.
  - **L** — If the file is a symbolic link (UNIX systems).
  - **P** — If the file is a pipe file (UNIX systems).
  - **R** — If the file is readable.
  - **W** — If the file is writable.
- The FULL-PATHNAME attribute returns the absolute pathname of the file specified in the FILE-NAME attribute.
- If the FILE-NAME attribute contains a simple filename or relative pathname, the PATHNAME attribute contains the pathname of the specified file starting with the directory on the PROPATH where it is found. Otherwise, the PATHNAME attribute contains the absolute pathname specified in the FILE-NAME attribute.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

FOCUS system handle

A handle to the field-level widget that is the current field.

Note: Does not apply to SpeedScript programming.

Syntax

```
FOCUS [ :attribute ]
```

attribute

An attribute of the widget that has current input focus.

Attributes

The specific attributes available depend on the type of the widget. You can determine the widget type by examining the FOCUS:TYPE attribute.

Example

The following example uses the FOCUS handle to provide helpful information to the user. The procedure displays an interface that contains several different types of widgets. If you type ?, the procedure displays a message specifying the type of widget that has focus and whether VALUE-CHANGED event is a valid event for that widget.

r-focus.p

```
DEFINE VARIABLE inv-price LIKE item.price.
DEFINE VARIABLE inv-value LIKE item.price.
DEFINE VARIABLE report-type AS INTEGER INITIAL 1.

DEFINE BUTTON ok-butt LABEL "OK" AUTO-GO.
DEFINE BUTTON cancel-butt LABEL "CANCEL" AUTO-ENDKEY.

FORM
   inv-price LABEL "Price"
      AT ROW 1.25 COLUMN 2
   report-type LABEL "Report Sorted ..."
      AT ROW 2.25 COLUMN 2
      VIEW-AS RADIO-SET RADIO-BUTTONS "By Catalog Page",   1,
                                      "By Inventory Value", 2
   SKIP
   ok-butt cancel-butt
   WITH FRAME select-frame SIDE-LABELS.

ON ? ANYWHERE
  DO:
     MESSAGE "This is a" FOCUS:TYPE + ". VALUE-CHANGED is"
         (IF VALID-EVENT(FOCUS, "VALUE-CHANGED") THEN "a" ELSE "NOT a")
         "valid event for this widget."
         VIEW-AS ALERT-BOX INFORMATION BUTTONS OK.
     RETURN NO-APPLY.
END.

ENABLE ALL WITH FRAME select-frame.

WAIT-FOR WINDOW-CLOSE OF CURRENT-WINDOW.
```

Note that this example prevents you from entering the question mark character (?) in any field. This does not cause a problem in `r-focus.p` because a question mark is not a valid input character for any field in the interface.

Notes

- A typical use of the FOCUS handle identifies the widget that contains the current text selection for reference by the system clipboard. For an example of this usage, see *OpenEdge Development: Programming Interfaces*.
- Within a WAIT-FOR statement, you can specify the field that receives initial input focus.
- You must give input focus to any fill-in widget where you want to set the AUTO-ZAP attribute. For more information, see the SAX-reader object handle reference entry.

See also SAX-reader object handle, WAIT-FOR statement

FONT-TABLE system handle (Windows only; Graphical interfaces only)

A handle to the current font table.

Note: Does not apply to SpeedScript programming.

Syntax

```
FONT-TABLE [ :attribute | :method ]
```

attribute

Specifies an attribute of the FONT-TABLE handle.

method

Specifies a method of the FONT-TABLE handle.

Attributes

NUM-ENTRIES attribute	TYPE attribute

Methods

GET-TEXT-HEIGHT-CHARS( ) method	GET-TEXT-HEIGHT-PIXELS( ) method
GET-TEXT-WIDTH-CHARS( ) method	GET-TEXT-WIDTH-PIXELS( ) method

Example

This code shows how to query and set the integer attribute, NUM-ENTRIES:

```
DEFINE VARIABLE i AS INTEGER.
i = FONT-TABLE:NUM-ENTRIES.    /* to query */
/* or */
i = 255.
FONT-TABLE:NUM-ENTRIES = i.    /* to set */
```

Notes

- Unlike the COLOR-TABLE system handle, the FONT-TABLE system handle does not allow you to set fonts dynamically. Font entries can only be changed by the user through the font system dialog box. Fonts are always dynamic.
- The current font table is the font table in the current environment, which is the startup environment or the environment most recently specified in a USE statement.
- To determine the number of font entries in the font table, query the NUM-ENTRIES attribute.
- To change the number of font entries in the font table, set the NUM-ENTRIES attribute.
- To allow users to set dynamic font table entries at run time, an application can display a font common dialog with the SYSTEM-DIALOG FONT statement.
- To save font definitions from the font table to the current environment file, use the PUT-KEY-VALUE statement. To retrieve the font definition specified in the current environment file, use the GET-KEY-VALUE statement.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

See also

GET-KEY-VALUE statement, PUT-KEY-VALUE statement, SYSTEM-DIALOG FONT statement, USE statement

LAST-EVENT system handle

A handle to the last event the application received.

Syntax

```
LAST-EVENT [ :attribute ]
```

attribute

An attribute of the LAST-EVENT.

Attributes

CODE attribute	COLUMN attribute	EVENT-PROCEDURE-CONTEXT attribute
FUNCTION attribute	LABEL attribute	ON-FRAME-BORDER attribute
ROW attribute	TYPE attribute	WIDGET-ENTER attribute
WIDGET-LEAVE attribute	X attribute	Y attribute

Example

This procedure creates a variety of widgets and a frame that acts as a message area. As you move around the widgets the procedure tells you what events Progress generates.

r-lstevt.p

```
DEFINE VARIABLE msg_watcher AS CHARACTER FORMAT "x(50)" NO-UNDO.
DEFINE VARIABLE fi AS CHARACTER FORMAT "x(50)" LABEL "Fill-in" NO-UNDO.
DEFINE VARIABLE edit AS CHARACTER VIEW-AS EDITOR SIZE 10 BY 2 NO-UNDO.
DEFINE BUTTON exit AUTO-GO LABEL "Go to Exit".
DEFINE BUTTON test LABEL "Test" SIZE 15 BY 3.

DEFINE VARIABLE txt0 AS CHARACTER FORMAT "x(75)" NO-UNDO.
DISPLAY "Feel free to move around..." VIEW-AS TEXT
     SKIP fi SKIP (0.1)
     edit AT 2 SKIP (0.1)
     test AT 10
     exit AT 50 SKIP (0.5)
   WITH FRAME f SIDE-LABELS.

DISPLAY txt0 LABEL "LAST-EVENT:LABEL FUNCTION (TYPE) > SELF:TYPE LABEL"
WITH FRAME report 4 DOWN CENTERED TITLE "Messages".
ON RETURN, TAB, ANY-PRINTABLE, GO ANYWHERE
   RUN msgwatch. /* This procedure uses LAST-EVENT. */

ENABLE ALL WITH FRAME f.
WAIT-FOR GO OF FRAME f FOCUS fi.
PROCEDURE msgwatch.
   txt0 = LAST-EVENT:LABEL + " " + LAST-EVENT:FUNCTION + " [" +
          LAST-EVENT:EVENT-TYPE + "]-->" + SELF:TYPE + " ".
   IF CAN-QUERY(SELF, "LABEL")
   THEN txt0 = txt0 + SELF:LABEL.
   ELSE txt0 = txt0 + STRING(SELF).

   DISPLAY txt0 WITH FRAME report.
   DOWN WITH FRAME report.
END PROCEDURE.
```

Notes

- For keyboard events, the CODE, FUNCTION, and LABEL attributes return the key code, key function, and key label of the event, respectively. For all other events the CODE attribute returns the numeric event code.

  For mouse events, the FUNCTION attribute returns the names of portable mouse events and the LABEL attribute returns the names of three-button mouse events.

  For high-level Progress events, the FUNCTION attribute returns the name of the event. If the Progress event is triggered by a key press, the LABEL attribute returns the key label. Otherwise, it returns the event name, as with the FUNCTION attribute.

- The EVENT-TYPE attribute returns the category of the event: KEYPRESS, MOUSE, or Progress.
- The ON-FRAME-BORDER attribute indicates whether a MOUSE event occurred in the border of a frame.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.
- The X and Y attributes return the pixel position of a MOUSE event relative to the current frame.
- For browse widgets, WIDGET-ENTER and WIDGET-LEAVE are different depending on whether the browse is editable or read-only. For editable browse widgets, WIDGET-ENTER contains the widget handle of the column with focus. For read-only browse widgets, WIDGET-ENTER contains the widget handle of the browse. For editable brows widgets, WIDGET-LEAVE contains the widget handle of the column the user just left. For read-only browse widgets, WIDGET-LEAVE contains the widget handle of the field-level widget the user just left.

See also LIST-EVENTS function, LIST-WIDGETS function, SAX-reader object handle, VALID-EVENT function

LOG-MANAGER system handle

A handle to logging settings for the current OpenEdge session.

Syntax

```
LOG-MANAGER [ :attribute | :method ]
```

attribute

An attribute of the LOG-MANAGER system handle.

method

A method of the LOG-MANAGER system handle.

Attributes

ENTRY-TYPES-LIST attribute	LOG-ENTRY-TYPES attribute	LOGFILE-NAME attribute
LOGGING-LEVEL attribute	LOG-THRESHOLD attribute	NUM-LOG-FILES attribute
TYPE attribute		

Methods

CLEAR-LOG( ) method	CLOSE-LOG( ) method
WRITE-MESSAGE( ) method	

For more information about logging, see *OpenEdge Development: Debugging and Troubleshooting*.

ProDataSet object handle

A handle to a ProDataSet object. A ProDataSet object is a collection of one or more related temp-tables. Each temp-table in a ProDataSet object can attach to a data-source object that allows filling of the temp-table from the data source, or updating the data source from the temp-table. A ProDataSet object can optionally contain a set of data relations between the temp-tables.

A ProDataSet object can be static or dynamic. A static ProDataSet object is one you define at compile time with the DEFINE DATASET statement. A dynamic ProDataSet object is one you create at run time with the CREATE DATASET statement. Use this handle to access the ProDataSet object's attributes and methods, and its sub-elements (its temp-table buffers, data relations, data sources, and so on) and their attributes and methods.

Syntax

```
dataset-object-handle [ :attribute | :method ]
```

dataset-object-handle

An item of type HANDLE representing a handle to a ProDataSet object.

attribute

An attribute of the ProDataSet object.

method

A method of the ProDataSet object.

Attributes

(1 of 2)

ADM-DATA attribute	DATA-SOURCE-MODIFIED attribute	DYNAMIC attribute
ERROR attribute	HANDLE attribute	INSTANTIATING-PROCEDURE attribute
NAME attribute	NAMESPACE-PREFIX attribute	NAMESPACE-URI attribute
NEXT-SIBLING attribute	NUM-BUFFERS attribute	NUM-REFERENCES attribute
NUM-RELATIONS attribute	NUM-TOP-BUFFERS attribute	PRIVATE-DATA attribute

(2 of 2)

REJECTED attribute	RELATIONS-ACTIVE attribute	TYPE attribute
UNIQUE-ID attribute		

Methods

ACCEPT-CHANGES( ) method	ADD-BUFFER( ) method
ADD-RELATION( ) method	APPLY-CALLBACK( ) method
CLEAR( ) method	COPY-DATASET( ) method
COPY-TEMP-TABLE( ) method	CREATE-LIKE( ) method
EMPTY-DATASET( ) method	FILL( ) method
GET-BUFFER-HANDLE( ) method	GET-CALLBACK-PROC-CONTEXT( ) method
GET-CALLBACK-PROC-NAME( ) method	GET-CHANGES( ) method
GET-RELATION( ) method	GET-TOP-BUFFER( ) method
MERGE-CHANGES( ) method	READ-XML( ) method
READ-XMLSCHEMA( ) method	REJECT-CHANGES( ) method
SET-BUFFERS( ) method	SET-CALLBACK( ) method
SET-CALLBACK-PROCEDURE( ) method	WRITE-XML( ) method
WRITE-XMLSCHEMA( ) method	

Events

AFTER-FILL event	BEFORE-FILL event

For information on these FILL events, see the "ProDataSet events" section on page 2195.

Note For information about dynamically accessing the data in a ProDataSet object, see the reference entries related to the TEMP-TABLE, BUFFER, BUFFER-FIELD, and QUERY objects.

See also Buffer object handle, CREATE DATASET statement, DEFINE DATASET statement, EMPTY-TEMP-TABLE( ) method, Temp-table object handle

Query object handle

A handle to a query object. A query object corresponds to an underlying Progress query, which can be static or dynamic. An example of a static underlying query is one you define at compile time with the DEFINE QUERY statement. An example of a dynamic underlying query is one you create at run time with the new CREATE QUERY statement.

Syntax

```
query-handle [ :attribute | :method ]
```

query-handle

An item of type WIDGET-HANDLE representing a handle to a query object.

attribute

An attribute of the query object.

method

A method of the query object.

Attributes

ADM-DATA attribute	BASIC-LOGGING attribute	CACHE attribute
CURRENT-RESULT-ROW attribute	DYNAMIC attribute	FORWARD-ONLY attribute
HANDLE attribute	INDEX-INFORMATION attribute	IS-OPEN attribute
NAME attribute	NUM-BUFFERS attribute	NUM-RESULTS attribute
PREPARE-STRING attribute	PRIVATE-DATA attribute	QUERY-OFF-END attribute
SKIP-DELETED-RECORD attribute	TYPE attribute	UNIQUE-ID attribute

Methods

ADD-BUFFER( ) method	APPLY-CALLBACK( ) method
CREATE-RESULT-LIST-ENTRY( ) method	DELETE-RESULT-LIST-ENTRY( ) method
DUMP-LOGGING-NOW( ) method	GET-BUFFER-HANDLE( ) method
GET-CALLBACK-PROC-CONTEXT( ) method	GET-CALLBACK-PROC-NAME( ) method
GET-CURRENT( ) method	GET-FIRST( ) method
GET-LAST( ) method	GET-NEXT( ) method
GET-PREV( ) method	QUERY-CLOSE( ) method
QUERY-OPEN( ) method	QUERY-PREPARE( ) method
REPOSITION-BACKWARD( ) method	REPOSITION-FORWARD( ) method
REPOSITION-TO-ROW( ) method	REPOSITION-TO-ROWID( ) method
SET-BUFFERS( ) method	SET-CALLBACK( ) method
SET-CALLBACK-PROCEDURE( ) method	

Note For more information on the query object, see *OpenEdge Development: Progress 4GL Handbook*.

See also Buffer object handle, Buffer-field object handle

RCODE-INFO system handle

A handle to a specific Progress r-code file.

```
RCODE-INFO [ :attribute ]
```

attribute

Specifies an attribute of the RCODE-INFO handle.

Attributes

CODEPAGE attribute	CRC-VALUE attribute	DB-REFERENCES attribute
FILE-NAME attribute	LANGUAGES attribute	MD5-VALUE attribute
TABLE-CRC-LIST attribute	TABLE-LIST attribute	TYPE attribute

Example

The following example prompts for the name of an r-code file and returns its CRC code and the languages for which it is compiled:

r-rcode.p

```
DEFINE VARIABLE rcode-file  AS CHARACTER FORMAT "x(60)" LABEL "File".

REPEAT:
  SET rcode-file WITH FRAME rc-info.
  RCODE-INFO:FILE-NAME = rcode-file.
  DISPLAY RCODE-INFO:CRC-VALUE LABEL "CRC"
    RCODE-INFO:LANGUAGES FORMAT "x(60)" LABEL "Languages"
    WITH FRAME rc-info SIDE-LABELS TITLE "R-code Check".
END.
```

Notes

- Progress generates an r-code file when you compile a procedure with the SAVE option of the COMPILE statement. You **cannot** use the RCODE-INFO handle to get information on session compiles.
- To use the RCODE-INFO handle, you must first set the FILE-NAME attribute to the name of an r-code file (with or without a `.r` or `.p` extension). If you do not provide a full pathname, Progress searches your `PROPATH` to find the file. You can then read the CRC-VALUE attribute and LANGUAGES attribute to get information on the file. If the r-code file is not found, both LANGUAGES and CRC-VALUE are set to the Unknown value (`?`).
- The LANGUAGES attribute holds a comma-separated list of language names supported by the r-code. The default segment appears in the list as the value `<unnamed>`.
- The CRC-VALUE attribute returns the r-code CRC value stored in the r-code. The calculation for this value is based on the filename and contents of the procedure file during compilation. This value is different from any database CRCs that are stored in the r-code. For more information on CRCs, see *OpenEdge Development: Progress 4GL Handbook*.
- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

SAX-attributes object handle

Contains the attribute values of an XML element in a SAX-reader object. The purpose of the object is to hold values needed by the SAX parser to set the attribute values of a new XML element as it adds the element to a SAX-reader object. As such, the object is automatically created when the SAX-reader PARSE( ) method calls the START-ELEMENT callback procedure. The SAX-attributes object is destroyed when the START-ELEMENT callback completes.

Because the SAX-attributes object exists for a short scope, you cannot access the object before or after the life-cycle of the START-ELEMENT callback. For this reason, you should create data elements with a longer scope if you need to save values from a SAX-attributes object.

Syntax

```
sax-attributes-handle [ :attribute | :method ]
```

sax-attributes-handle

A variable of type HANDLE.

attribute

An attribute of the SAX-attributes object.

method

A method of the SAX-attributes object.

Attributes

ADM-DATA attribute	NUM-ITEMS attribute	PRIVATE-DATA attribute
TYPE attribute	UNIQUE-ID attribute	

Methods

GET-INDEX-BY-NAMESPACE-NAME( ) method	GET-INDEX-BY-QNAME( ) method
GET-LOCALNAME-BY-INDEX( ) method	GET-QNAME-BY-INDEX( ) method
GET-TYPE-BY-INDEX( ) method	GET-TYPE-BY-NAMESPACE-NAME( ) method
GET-TYPE-BY-QNAME( ) method	GET-URI-BY-INDEX( ) method
GET-VALUE-BY-INDEX( ) method	GET-VALUE-BY-NAMESPACE-NAME( ) method
GET-VALUE-BY-QNAME( ) method	

See also CREATE SAX-READER statement, SAX-reader object handle

SAX-reader object handle

Provides access to the SAX parser to stream an XML document into the object. The SAX-reader object, used with the SAX interface, corresponds to the X-document object of the Document Object Model (DOM) interface, but presents a very different programming interface.

Syntax

```
sax-reader-handle [ :attribute | :method ]
```

sax-reader-handle

A variable of type HANDLE.

attribute

An attribute of the SAX-reader object.

method

A method of the SAX-reader object.

Attributes

ADM-DATA attribute	HANDLER attribute	LOCATOR-COLUMN-NUMBER attribute
LOCATOR-LINE-NUMBER attribute	LOCATOR-PUBLIC-ID attribute	LOCATOR-SYSTEM-ID attribute
NONAMESPACE-SCHEMA-LOCATION attribute	PARSE-STATUS attribute	PRIVATE-DATA attribute
SCHEMA-LOCATION attribute	SCHEMA-PATH attribute	SUPPRESS-NAMESPACE-PROCESSING attribute
TYPE attribute	UNIQUE-ID attribute	VALIDATION-ENABLED attribute

Methods

ADD-SCHEMA-LOCATION( ) method	SAX-PARSE( ) method
SAX-PARSE-FIRST( ) method	SAX-PARSE-NEXT( ) method
SET-INPUT-SOURCE( ) method	STOP-PARSING( ) method

See also

CREATE SAX-READER statement, SAX-attributes object handle

SAX-writer object handle

A handle to the SAX-writer object. You create the handle and assign it to a handle variable with the CREATE SAX-WRITER statement.

Syntax

```
handle [ :attribute | :method ]
```

handle

A variable of type HANDLE.

attribute

An attribute of the SAX-writer object.

method

A method of the SAX-writer object.

Attributes

ENCODING attribute	FORMATTED attribute	FRAGMENT attribute
STANDALONE attribute	STRICT attribute	VERSION attribute
WRITE-STATUS attribute		

Methods

DECLARE-NAMESPACE( ) method	END-DOCUMENT( ) method
END-ELEMENT( ) method	INSERT-ATTRIBUTE( ) method
RESET( ) method	SET-OUTPUT-DESTINATION( ) method
START-DOCUMENT( ) method	START-ELEMENT( ) method
WRITE-CDATA( ) method	WRITE-CHARACTERS( ) method
WRITE-COMMENT( ) method	WRITE-DATA-ELEMENT( ) method
WRITE-EMPTY-ELEMENT( ) method	WRITE-ENTITY-REF( ) method
WRITE-EXTERNAL-DTD( ) method	WRITE-FRAGMENT( ) method
WRITE-PROCESSING-INSTRUCTION( ) method	

See also

CREATE SAX-WRITER statement

SECURITY-POLICY system handle

A handle to the security policy settings for the current OpenEdge session, including data cryptography, user authentication, and trusted authentication domain registry maintenance.

Syntax

```
SECURITY-POLICY [ :attribute | :method ]
```

attribute

An attribute of the SECURITY-POLICY handle.

method

A method of the SECURITY-POLICY handle.

Attributes

ENCRYPTION-SALT attribute	PBE-HASH-ALGORITHM attribute
PBE-KEY-ROUNDS attribute	SYMMETRIC-ENCRYPTION-ALGORITHM attribute
SYMMETRIC-ENCRYPTION-IV attribute	SYMMETRIC-ENCRYPTION-KEY attribute
SYMMETRIC-SUPPORT attribute	

Methods

LOAD-DOMAINS( ) method	LOCK-REGISTRATION( ) method
REGISTER-DOMAIN( ) method	SET-CLIENT( ) method

See also

Client-principal object handle, DECRYPT function, ENCRYPT function, GENERATE-PBE-KEY function, GENERATE-PBE-SALT function, GENERATE-RANDOM-KEY function, MD5-DIGEST function, SHA1-DIGEST function

SELF system handle

A handle to the object or widget associated with the currently executing user-interface trigger or event procedure.

Syntax

```
SELF [ :attribute ]
```

attribute

An attribute of the object or widget associated with the trigger or event procedure.

Attributes

The specific attributes available depend on the type of the object or widget. You can determine the object or widget type by examining the SELF:TYPE attribute.

Example The following example uses the SELF handle to display the starting and ending positions of an object you move:

r-self.p

```
DEFINE BUTTON b_quit LABEL "Quit"
  TRIGGERS:
    ON CHOOSE QUIT.
  END.

DEFINE VARIABLE x AS CHAR INIT "MOVE ME".

DEFINE FRAME move
  x NO-LABEL
  WITH SIZE 80 BY 10 TITLE "Move/Resize Widget".

ASSIGN x:MOVABLE = TRUE
       x:SELECTABLE = TRUE.

DEFINE FRAME butt-frame
  b_quit
  WITH CENTERED ROW SCREEN-LINES - 1.

ON END-MOVE OF x IN FRAME move
  DISPLAY
    SELF:FRAME-ROW
    SELF:FRAME-COL
    WITH FRAME end-info CENTERED ROW 14 TITLE "End Position".

ON START-MOVE OF x IN FRAME move
  DISPLAY
    SELF:FRAME-ROW
    SELF:FRAME-COL
    WITH FRAME info CENTERED ROW 12 TITLE "Start Position".

ENABLE b_quit WITH FRAME butt-frame.
DISPLAY x WITH FRAME move.
ENABLE x WITH FRAME move.

WAIT-FOR CHOOSE OF b_quit IN FRAME butt-frame FOCUS x.
```

Notes

- You can reference the SELF handle only within a user-interface trigger or the event procedure for an ActiveX control or asynchronous remote request.
- In user-interface triggers, SELF is not automatically the widget that has input focus. To give input focus to the widget referenced by SELF, you must apply the ENTRY event to SELF within the trigger block. Note that you must do this for fill-in widgets whose AUTO-ZAP attribute you want to set, as in this fragment:

```
DEFINE VARIABLE fname AS CHARACTER FORMAT "x(30)" LABEL "Name".
DEFINE FRAME FillFrame fname WITH SIDE-LABELS.

ON ENTRY OF fname IN FRAME FillFrame DO:
    APPLY "ENTRY" TO SELF.
    SELF:AUTO-ZAP = TRUE.
END.
```

  This makes SELF = FOCUS, which allows the new AUTO-ZAP value to take effect. For more information on the AUTO-ZAP attribute, see the "Attributes and Methods Reference" section on page 1497.
- In the event procedure of an asynchronous remote request or in the context of a procedure called directly or indirectly by this event procedure, SELF returns the associated asynchronous request handle.
- In the event procedure of an ActiveX control, SELF returns the control-frame widget handle and the COM-SELF system handle returns the control-frame COM-HANDLE value.
- If referenced within a READ-RESPONSE event procedure, then SELF is the socket handle associated with the connection that received the message. If referenced within the CONNECT event procedure, then SELF is the server socket handle.

See also

Asynchronous request object handle, COM-SELF system handle, FOCUS system handle, LAST-EVENT system handle

Server object handle

Allows you to connect and execute remote procedures on an AppServer or Web service.

Note: This handle does not provide direct access to an AppServer session context as does a SESSION handle for the current context. Rather, it provides access to a server object in the current context that allows you to connect, disconnect, and retrieve a variety of information on a connected AppServer.

Syntax

```
server-handle [ :attribute | :method ]
```

server-handle

A handle variable that references a server object created by the CREATE SERVER statement that, in turn, allows you to connect to and access an AppServer instance or a Web service application.

attribute

An attribute of the server handle.

method

A method of the server handle.

Attributes

ASYNC-REQUEST-COUNT attribute	CLIENT-CONNECTION-ID attribute[1]	FIRST-ASYNC-REQUEST attribute
FIRST-PROCEDURE attribute[1]	LAST-ASYNC-REQUEST attribute	LAST-PROCEDURE attribute[1]
NAME attribute	NEXT-SIBLING attribute	PREV-SIBLING attribute
PRIVATE-DATA attribute	SSL-SERVER-NAME attribute	SUBTYPE attribute
TYPE attribute		

[1] For this attribute to be valid, the handle must have an AppServer or Web service connection (the CONNECTED( ) method must return TRUE).

Methods

CANCEL-REQUESTS( ) method	CONNECT( ) method (AppServer)
CONNECT( ) method (Web service)	CONNECTED( ) method
DISCONNECT( ) method	

Note For SpeedScript, as in any 4GL client, a WebSpeed Agent can use a valid server handle to access and run remote procedures on an AppServer. However, it does not access or affect the state of any WebSpeed Transaction Server.

Server socket object handle

A handle to a server socket object. This object allows you to listen for and accept TCP/IP socket connections on a given port.

Note: Does not apply to SpeedScript programming.

Syntax

```
server-socket-handle [ :attribute | :method ]
```

server-socket-handle

A handle variable that references a server socket object created by the CREATE SERVER-SOCKET statement that, in turn, allows you to listen for and accept multiple connections on a given port.

attribute

An attribute of the server socket handle.

method

A method of the server socket handle.

Attributes

HANDLE attribute	NAME attribute	NEXT-SIBLING attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	SENSITIVE attribute
TYPE attribute		

Methods

DISABLE-CONNECTIONS( ) method	ENABLE-CONNECTIONS( ) method
SET-CONNECT-PROCEDURE( ) method	

Events

CONNECT event

Note The server socket object is used to enable Progress to listen to and accept new connections from socket clients; it is via the socket object that clients and servers communicate. For more information on using sockets, see *OpenEdge Development: Programming Interfaces*.

SESSION system handle

A handle to the current OpenEdge session object. This object allows you to read and modify the current OpenEdge session context.

Syntax

```
SESSION [ :attribute | :method ]
```

attribute

Specifies an attribute of the SESSION system handle.

method

Specifies a method of the SESSION system handle.

Attributes

(1 of 2)

APPL-ALERT-BOXES attribute	BASE-ADE attribute
BATCH-MODE attribute	CHARSET attribute
CLIENT-TYPE attribute	CONTEXT-HELP-FILE attribute
CPCASE attribute	CPCOLL attribute
CPINTERNAL attribute	CPLOG attribute
CPPRINT attribute	CPRCODEIN attribute
CPRCODEOUT attribute	CPSTREAM attribute
CPTERM attribute	DATA-ENTRY-RETURN attribute
DATE-FORMAT attribute	DEBUG-ALERT attribute
DISPLAY-TIMEZONE attribute	DISPLAY-TYPE attribute
FIRST-BUFFER attribute	FIRST-CHILD attribute
FIRST-DATASET attribute	FIRST-DATA-SOURCE attribute
FIRST-OBJECT attribute	FIRST-PROCEDURE attribute
FIRST-QUERY attribute	FIRST-SERVER attribute
FIRST-SERVER-SOCKET attribute	FIRST-SOCKET attribute
FRAME-SPACING attribute	HEIGHT-CHARS attribute
HEIGHT-PIXELS attribute	ICFPARAMETER attribute

(2 of 2)

IMMEDIATE-DISPLAY attribute	LAST-CHILD attribute
LAST-OBJECT attribute	LAST-PROCEDURE attribute
LAST-SERVER attribute	LAST-SERVER-SOCKET attribute
LAST-SOCKET attribute	MULTITASKING-INTERVAL attribute
NEXT-SIBLING attribute	NUMERIC-DECIMAL-POINT attribute
NUMERIC-FORMAT attribute	NUMERIC-SEPARATOR attribute
PARAMETER attribute	PIXELS-PER-COLUMN attribute
PIXELS-PER-ROW attribute	PRINTER-CONTROL-HANDLE attribute
PRINTER-HDC attribute	PRINTER-NAME attribute
PRINTER-PORT attribute	PROXY-PASSWORD attribute
PROXY-USERID attribute	REMOTE attribute
SCHEMA-CHANGE attribute	SERVER-CONNECTION-BOUND attribute
SERVER-CONNECTION-BOUND-REQUEST attribute	SERVER-CONNECTION-CONTEXT attribute
SERVER-CONNECTION-ID attribute	SERVER-OPERATING-MODE attribute
STARTUP-PARAMETERS attribute	STREAM attribute
SUPER-PROCEDURES attribute	SUPPRESS-WARNINGS attribute
SYSTEM-ALERT-BOXES attribute	TEMP-DIRECTORY attribute
THREE-D attribute	TIME-SOURCE attribute
TOOLTIPS attribute	TYPE attribute
V6DISPLAY attribute	WIDTH-CHARS attribute
WIDTH-PIXELS attribute	WINDOW-SYSTEM attribute
WORK-AREA-HEIGHT-PIXELS attribute	WORK-AREA-WIDTH-PIXELS attribute
WORK-AREA-X attribute	WORK-AREA-Y attribute
YEAR-OFFSET attribute	

Methods

ADD-SUPER-PROCEDURE( ) method	EXPORT( ) method
GET-PRINTERS( ) method	GET-WAIT-STATE( ) method
REMOVE-SUPER-PROCEDURE( ) method	SET-NUMERIC-FORMAT( ) method
SET-WAIT-STATE( ) method	

Example

The following example uses the SESSION:IMMEDIATE-DISPLAY attribute. When dumping or loading records from the database, the procedure displays a running count of records. If IMMEDIATE-DISPLAY is false, no value is shown until all records are dumped or loaded. At that point, the total is shown. To prevent this, IMMEDIATE-DISPLAY is set to true just before the dump or load and then reset to false afterwards.

r-dstrig.p *(1 of 2)*

```
DEFINE SUB-MENU file
  MENU-ITEM viewit LABEL "&View Data"
  MENU-ITEM dumpit LABEL "&Dump Data"
  MENU-ITEM loadit LABEL "&Load Data".
  MENU-ITEM exit   LABEL "E&xit".
  DEFINE MENU mbar MENUBAR
  SUB-MENU file LABEL "&File".
  DEFINE BUTTON b_more LABEL "Next".
DEFINE BUTTON b_exit LABEL "Cancel".
DEFINE FRAME cust-frame
  customer.cust-num SKIP
  customer.name  SKIP
  customer.phone SKIP
  b_more b_exit
  WITH CENTERED SIDE-LABELS ROW 3.
DEFINE STREAM cust.
DEFINE VARIABLE i AS INTEGER NO-UNDO.

PAUSE 0 BEFORE-HIDE.

ON CHOOSE OF b_exit IN FRAME cust-frame
DO:
  HIDE FRAME cust-frame NO-PAUSE.
  DISABLE ALL WITH FRAME cust-frame.
  LEAVE.
END.

ON CHOOSE OF b_more IN FRAME cust-frame
DO:
  FIND NEXT customer NO-LOCK NO-ERROR.
  IF NOT AVAILABLE(customer) THEN
    RETURN.
  DISPLAY customer.cust-num customer.name customer.phone
    WITH FRAME cust-frame.
END.
```

r-dstrig.p *(2 of 2)*

```
ON CHOOSE OF MENU-ITEM viewit
DO:
  ENABLE ALL WITH FRAME cust-frame.
  FIND FIRST customer NO-LOCK NO-ERROR.
  DISP customer.cust-num customer.name customer.phone
    WITH FRAME cust-frame.
  APPLY "ENTRY" TO b_more.
END.

ON CHOOSE OF MENU-ITEM dumpit
DO:
  DISABLE TRIGGERS FOR DUMP OF customer.
  i = 1.
  SESSION:IMMEDIATE-DISPLAY = TRUE.
  OUTPUT STREAM cust TO "customer.d".
  FOR EACH customer NO-LOCK:
    EXPORT STREAM cust customer.
    DISP i LABEL "Records Processed"
      WITH FRAME rec-info SIDE-LABELS ROW SCREEN-LINES / 2 CENTERED.
    i = i + 1.
  END.
  SESSION:IMMEDIATE-DISPLAY = FALSE.
  OUTPUT STREAM cust CLOSE.
END.

ON CHOOSE OF MENU-ITEM loadit
DO:
  DISABLE TRIGGERS FOR LOAD OF customer.
  INPUT FROM "customer.d".
  SESSION:IMMEDIATE-DISPLAY = TRUE.
  REPEAT:
    CREATE customer.
    IMPORT customer.
    DISP i LABEL "Records Processed"
      WITH FRAME rec-info SIDE-LABELS ROW SCREEN-LINES / 2 CENTERED.
    i = i + 1.
  END.
  INPUT CLOSE.
  SESSION:IMMEDIATE-DISPLAY = FALSE.
END.

IF NOT RETRY THEN
ASSIGN CURRENT-WINDOW:MENUBAR = MENU mbar:HANDLE
       CURRENT-WINDOW:VISIBLE = TRUE.

WAIT-FOR CHOOSE OF MENU-ITEM exit.
```

Notes

- Several attributes of the SESSION handle control the execution of Progress code during the current OpenEdge session. This means that the SESSION handle controls the behavior of any code that you are developing and testing, and the OpenEdge ADE toolset. While the tools of the OpenEdge ADE monitor and set the attributes of the SESSION handle to meet their needs, it is possible that the execution of a procedure that sets attributes of the SESSION handle may affect the display and behavior of the OpenEdge ADE toolset.

- The FIRST-PROCEDURE and LAST-PROCEDURE attributes are set or reset when you create or delete the first or last persistent procedure in a session. You can use procedure attributes to navigate the procedure entries, reference information, and manage the user interface for each persistent procedure in the procedure chain accessed by FIRST-PROCEDURE and LAST-PROCEDURE.

  For more information on the attributes of procedure handles, see the THIS-PROCEDURE system handle reference entry. For information on creating a persistent procedure, see the RUN statement reference entry. For information on deleting a persistent procedure, see the DELETE PROCEDURE statement reference entry.

- The FIRST-SERVER and LAST-SERVER attributes are set or reset when you create or delete the first or last server handle in a session. You can use server handle attributes and methods to navigate the current chain of server handles, connect to a running AppServer, reference information on a connected AppServer, access remote persistent procedures running on a connected AppServer, and disconnect from a connected AppServer for each server handle in the chain accessed by FIRST-SERVER and LAST-SERVER.

  For more information on the attributes and methods of server handles, see the Server object handle reference entry. For information on creating server handles, see the CREATE SAX-READER statement reference entry.

- Setting the IMMEDIATE-DISPLAY attribute to TRUE can significantly slow performance. However, some code segments may not execute properly with IMMEDIATE-DISPLAY set to FALSE. If a segment of code requires that IMMEDIATE-DISPLAY is TRUE, you should set the attribute to TRUE immediately before the code segment and change it back to FALSE immediately after the segment.

- In Windows, when execution is blocked for input (by a WAIT-FOR statement, for example), Progress listens for messages from the windowing system. This allows Progress to multitask properly with other Windows applications. However, if your OpenEdge application performs long processing without blocking for input, then it may not multitask properly because Progress does not automatically check for messages from the windowing system. To force Progress to poll for windowing system messages during this time, you can set the MULTITASKING-INTERVAL attribute to a non-zero value. The lower the value, the more often Progress checks for messages. This may decrease Progress performance. The maximum value is 9999. A value of 0 inhibits polling until Progress blocks for input.

  If you set MULTITASKING-INTERVAL to a non-zero value for a code segment, reset it to 0 immediately after that code.

- Progress sets the TEMP-DIRECTORY attribute to the value you specify for the Temporary Directory (-T) parameter. If you omit the -T parameter, TEMP-DIRECTORY is set to your current working directory.

- The TYPE attribute returns the widget type, PSEUDO-WIDGET.

- Use the SET-WAIT-STATE method to prevent user and system input, and provide visual feedback during a long computation or other background process. The value you pass determines the type of wait message or cursor the windowing system displays for the user. Passing the value "" to SET-WAIT-STATE ends the wait state. Use this method only for long computations or other processes that force the user to wait significantly longer than the usual response time.

- If you set a wait state for your application, Progress automatically ends the wait state if it displays an alert box, a dialog box, or message update.

- For SpeedScript, the invalid attributes are: APPL-ALERT-BOXES, CONTEXT-HELP-FILE, DATA-ENTRY-RETURN, FIRST-CHILD, HEIGHT-PIXELS, LAST-CHILD, PARAMETER, PIXELS-PER-COLUMN, PIXELS-PER-ROW, SUPPRESS-WARNINGS, SYSTEM-ALERT-BOXES, THREE-D, TOOLTIPS, V6DISPLAY, WIDTH-PIXELS. The GET-PRINTERS( ) method is invalid for SpeedScript.

SOAP-fault object handle

A handle to a SOAP-fault object. A SOAP-fault object contains information specific to a SOAP fault.

Note: Does not apply to SpeedScript programming.

Syntax

```
soap-fault-handle [ :attribute ]
```

soap-fault-handle

A handle variable that references a SOAP-fault object.

attribute

An attribute of the SOAP-fault object.

Attributes

SOAP-FAULT-ACTOR attribute	SOAP-FAULT-CODE attribute	SOAP-FAULT-DETAIL attribute
SOAP-FAULT-STRING attribute	TYPE attribute	

Notes

- When Progress detects a SOAP fault message, it converts the SOAP fault message to a Progress error message and creates a SOAP-fault object (identified by the ERROR-OBJECT-DETAIL attribute on the ERROR-STATUS system handle). A SOAP-fault object exists only as long as its related ERROR-STATUS entry (that is, until the execution of another statement containing the NO-ERROR option).
- Use the SOAP–FAULT-DETAIL object handle to access the SOAP fault message detail.

See also

ERROR-STATUS system handle, SOAP-fault-detail object handle

SOAP-fault-detail object handle

A handle to a SOAP-fault-detail object.

Note: Does not apply to SpeedScript programming.

Syntax

```
soap-fault-detail-handle [ :attribute | :method ]
```

soap-fault-detail-handle

A handle variable that references a SOAP-fault-detail object.

attribute

An attribute of the SOAP-fault-detail object.

method

A method of the SOAP–FAULT-DETAIL object.

Attributes

TYPE attribute

Methods

GET-NODE( ) method GET-SERIALIZED( ) method

Note

You can use the GET-NODE( ) method to get an X-noderef object handle that refers to the XML that underlies a SOAP-fault-detail object. The application can then use this X-noderef object handle to access the underlying XML. The only restriction is that the application cannot use the X-noderef object handle retrieved from the SOAP-fault-detail object to access the X-document associated with the SOAP-fault object. For more information, see *OpenEdge Development: Web Services*.

SOAP-header object handle

A handle to a SOAP-header object. A SOAP-header object is passed as an input parameter to a response callback procedure and as an output parameter to a request callback procedure.

Note: Does not apply to SpeedScript programming.

Syntax

```
soap-header-handle [ :attribute | :method ]
```

soap-header-handle

A handle variable that references a SOAP-header object.

attribute

An attribute of the SOAP-header object.

method

A method of the SOAP-header object.

Attributes

ADM-DATA attribute	NAME attribute	NUM-HEADER-ENTRIES attribute
PRIVATE-DATA attribute	TYPE attribute	UNIQUE-ID attribute

Methods

ADD-HEADER-ENTRY( ) method	GET-HEADER-ENTRY( ) method

Note

The SOAP-header object is either implicitly created by Progress or explicitly created by the application using the CREATE SOAP-HEADER statement. In either case, the application is responsible for deleting this object. Use the DELETE OBJECT statement to delete a SOAP-header object and its underlying XML.

See also

CREATE SOAP-HEADER statement, SOAP-header-entryref object handle

SOAP-header-entryref object handle

A handle to a SOAP-header-entryref object.

Note: Does not apply to SpeedScript programming.

Syntax

```
soap-header-entryref-handle [ :attribute | :method ]
```

soap-header-entryref-handle

A handle variable that references a SOAP-header-entryref object.

attribute

An attribute of the SOAP-header-entryref object.

method

A method of the SOAP-header-entryref object.

Attributes

ACTOR attribute	ADM-DATA attribute	LOCAL-NAME attribute
MUST-UNDERSTAND attribute	NAME attribute	NAMESPACE-URI attribute
PRIVATE-DATA attribute	TYPE attribute	UNIQUE-ID attribute

Methods

DELETE-HEADER-ENTRY( ) method	GET-NODE( ) method
GET-SERIALIZED( ) method	SET-ACTOR( ) method
SET-MUST-UNDERSTAND( ) method	SET-NODE( ) method
SET-SERIALIZED( ) method	

Notes

- The application is responsible for deleting this object. Use the DELETE OBJECT statement to delete a SOAP-header-entryref object without deleting its underlying XML. To delete the XML underlying the SOAP-header-entryref object, without deleting the object, use the DELETE-HEADER-ENTRY() method.
- You can use the GET-NODE( ) method to get an X-noderef object handle that refers to the XML that underlies a SOAP-header-entryref object. The application can then use this X-noderef object handle to access the underlying XML. The only restriction is that the application cannot use the X-noderef object handle retrieved from the SOAP-header-entryref object to access the X-document associated with the SOAP-header object. For more information, see *OpenEdge Development: Web Services*.

See also CREATE SOAP-HEADER-ENTRYREF statement, SOAP-header object handle

Socket object handle

A handle to a socket object. This object allows you to read or write data on a TCP/IP socket and to perform other TCP/IP socket actions.

Syntax

```
socket-handle [ :attribute | :method ]
```

socket-handle

A handle variable that references a socket object created by the CREATE SOCKET statement and that allows you to connect to, read from and write to a socket.

attribute

An attribute of the socket handle.

method

A method of the socket handle.

Attributes

BYTES-READ attribute	BYTES-WRITTEN attribute	HANDLE attribute
LOCAL-HOST attribute	LOCAL-PORT attribute	NAME attribute
NEXT-SIBLING attribute	PREV-SIBLING attribute	PRIVATE-DATA attribute
REMOTE-HOST attribute	REMOTE-PORT attribute	SENSITIVE attribute
SSL-SERVER-NAME attribute	TYPE attribute	

Methods

CONNECT( ) method	CONNECTED( ) method
DISCONNECT( ) method	GET-BYTES-AVAILABLE( ) method
GET-SOCKET-OPTION( ) method	READ( ) method
SET-READ-RESPONSE-PROCEDURE( ) method	SET-SOCKET-OPTION( ) method
WRITE( ) method	

Events

READ-RESPONSE event

Note The server socket object is used to enable connections from socket clients; it is via the socket object that clients and servers communicate. For more information on using sockets, see *OpenEdge Development: Programming Interfaces*.

SOURCE-PROCEDURE system handle

A handle to the procedure file that contains the original invocation (RUN statement or function invocation) of the current internal procedure or user-defined function.

Syntax

```
SOURCE-PROCEDURE [ :attribute | :method ]
```

attribute

An attribute of the SOURCE-PROCEDURE handle.

method

A method of the SOURCE-PROCEDURE handle.

Attributes

The SOURCE-PROCEDURE handle supports all the attributes of the procedure handle. For a list of these attributes, see the reference entry for the THIS-PROCEDURE system handle in this chapter.

Methods

The SOURCE-PROCEDURE handle supports all the methods of the procedure handle. For a list of these methods, see the reference entry for the THIS-PROCEDURE system handle in this chapter.

Examples

The following scenarios illustrate using SOURCE-PROCEDURE without procedure overriding, with procedure overriding, and with super and non-super RUNs:

Scenario 1: Without procedure overriding

The following scenario uses SOURCE-PROCEDURE without procedure overriding:

1. A and B are handles of procedure files running persistently.
2. proc1 is an internal procedure that resides in B.
3. A says “RUN proc1 IN B,” which runs B’s proc1.

In this scenario:

- The original run statement for proc1 occurs in Step 3.
- Within B’s proc1 (and within any proc1 that runs as a result of its original RUN statement), SOURCE-PROCEDURE is A.

Scenario 2: With procedure overriding

The following scenario uses SOURCE-PROCEDURE with procedure overriding:

1. A, B, and C, and X are handles of procedure files running persistently.
2. B is a super procedure of A, and C is a super procedure of B.
3. proc1 is an internal procedure different versions of which reside in A, B, and C.

   Note: This is an example of procedure overriding.

4. X says "RUN proc1 IN A," which runs A's proc1.
5. A's proc1 says "RUN SUPER," which runs B's proc1.
6. B's proc1 says "RUN SUPER," which runs C's proc1.

In this scenario:

- The original run statement for proc1 occurs in Step 4.
- Within any version of proc1 that runs as a result of its original RUN statement, SOURCE-PROCEDURE is X.

Scenario 3: With Super and Non-super RUNs

The following scenario shows how the value of SOURCE-PROCEDURE changes when a non-super RUN occurs:

1. A, B, and C are handles of procedure files running persistently.
2. B is a super procedure of A, and C is a super procedure of B.
3. proc1 is an internal procedure different versions of which reside in A, B, and C.
4. proc2 is an internal procedure different versions of which reside in A, B, and C.
5. A says "RUN proc1," which runs A's proc1.
6. A's proc1 says "RUN SUPER," which runs B's proc1.
7. B's proc1 says "RUN SUPER," which runs C's proc1.

Note: At this point, within any proc1 that runs as a result of its original RUN statement, the value of SOURCE-PROCEDURE is A.

8. C's proc1 says "RUN proc2," which runs C's proc2.

Note: This is a non-super RUN.

In this scenario:

- The original RUN statement for proc1 occurs in Step 5.
- Within any proc1 that runs as a result of its original RUN statement, SOURCE-PROCEDURE is A.
- The original RUN statement for proc2 occurs in Step 8.
- Within any proc2 that runs as a result of its original RUN statement, SOURCE-PROCEDURE is C.

For a sample program that uses SOURCE-PROCEDURE, see the reference entry for the RUN SUPER statement.

Notes

- You can use SOURCE-PROCEDURE in applications that do not use super procedures.
- In the main block of a procedure, the value of SOURCE-PROCEDURE is the handle of the procedure that ran the current 4GL source code or r-code file. This allows any Progress program to identify its caller, and to perform a "callback" to its caller.
- If a 4GL or other client runs a procedure on an AppServer, then in the procedure running on the AppServer, the value of SOURCE-PROCEDURE is the Unknown value (?).
- For more information on super procedures and procedure overriding, see *OpenEdge Development: Progress 4GL Handbook*.

See also

ADD-SUPER-PROCEDURE( ) method, REMOVE-SUPER-PROCEDURE( ) method, RUN SUPER statement, SUPER function, SUPER-PROCEDURES attribute, TARGET-PROCEDURE system handle

TARGET-PROCEDURE system handle

From within an internal procedure: A handle to the procedure file mentioned, explicitly or implicitly, by the original RUN statement that invoked (perhaps through a chain of super procedures) the current internal procedure.

From within a user-defined function: A handle to the procedure file mentioned, explicitly or implicitly, by the original function invocation that invoked (perhaps through a chain of super versions of functions) the current user-defined function.

Syntax

```
TARGET-PROCEDURE [ :attribute | :method ]
```

attribute

An attribute of the TARGET-PROCEDURE handle.

method

A method of the TARGET-PROCEDURE handle.

Attributes

The TARGET-PROCEDURE handle supports all the attributes of the procedure handle. For a list of these attributes, see the reference entry for the THIS-PROCEDURE system handle in this chapter.

Methods

The TARGET-PROCEDURE handle supports all the methods of the procedure handle. For a list of these methods, see the reference entry for the THIS-PROCEDURE system handle in this chapter.

Examples

The following scenarios illustrate using TARGET-PROCEDURE without procedure overriding, with procedure overriding, and with super and non-super RUNs:

Scenario 1: Without procedure overriding

The following scenario uses TARGET-PROCEDURE without procedure overriding:

1. A and B are handles of procedure files running persistently.
2. proc1 is an internal procedure that resides in B.
3. A says “RUN proc1 IN B,” which runs B’s proc1.

In this scenario:

- The original RUN statement for proc1 occurs in Step 3.
- Within proc1 (and any proc1 that runs as a result its original RUN statement), the value of TARGET-PROCEDURE is B.

Scenario 2: With procedure overriding

The following scenario uses TARGET-PROCEDURE with procedure overriding:

1. A, B, and C, and X are handles of procedure files running persistently.

2. B is a super procedure of A, and C is a super procedure of B.

3. proc1 is an internal procedure, different versions of which reside in A, B, and C.

Note: This is an example of procedure overriding.

4. X says "RUN proc1 in A," which runs A's proc1.

5. A's proc1 says "RUN SUPER," which runs B's proc1.

6. B's proc1 says "RUN SUPER," which runs C's proc1.

In this scenario:

- The original RUN statement for proc1 occurs in Step 4.
- Within any version of proc1 that runs as a result of the original RUN statement, the value of TARGET-PROCEDURE is A.

Scenario 3: With procedure overriding and additional complications

The following scenario uses TARGET-PROCEDURE with procedure overriding:

1. A, B, and C, and X are handles of procedure files running persistently.

2. B is a super procedure of A, and C is a super procedure of B.

3. proc1 is an internal procedure, different versions of which reside in B and C.

Note: proc1 does not reside in A.

4. X says “RUN proc1 in A,” which runs B’s proc1 (since A has no proc1 and B is a super procedure of A).

5. B’s proc1 says “RUN SUPER,” which runs C’s proc1.

In this scenario:

- The original RUN statement for proc1 occurs in Step 4.
- Within any version of proc1 that runs as a result of its original RUN statement, the value of TARGET-PROCEDURE is A.

Scenario 4: With Super and Non-super RUNs

The following scenario shows how the value of TARGET-PROCEDURE changes when a non-super RUN occurs:

1. A, B, and C are handles of procedure files running persistently.

2. B is a super procedure of A, and C is a super procedure of B.

3. proc1 is an internal procedure different versions of which reside in A, B, and C.

4. proc2 is an internal procedure different versions of which reside in A, B, and C.

5. A says “RUN proc1,” which runs A’s proc1.

6. A’s proc1 says “RUN SUPER,” which runs B’s proc1.

Note: At this point, within any version of proc1 that runs as a result of its original RUN statement, the value of TARGET-PROCEDURE is A.

7. B’s proc1 says “RUN proc2,” which runs B’s proc2.

Note: This is a non-super RUN.

In this scenario:

- The original RUN statement for proc2 occurs in Step 7.
- Within any proc2 that runs as a result of its original RUN statement, the value of TARGET-PROCEDURE is B.

For a sample program that uses TARGET-PROCEDURE, see the reference entry for the RUN SUPER statement in this book.

Notes

- You can use TARGET-PROCEDURE in applications that do not use super procedures.
- The value of TARGET-PROCEDURE becomes THIS-PROCEDURE in the following places:
  - Within the main block of a procedure file.
  - Within an internal procedure that is not a super version of another internal procedure.
  - Within a user-defined function that is not a super version of another user-defined function.
- For more information on super procedures, see *OpenEdge Development: Progress 4GL Handbook*.

See also

ADD-SUPER-PROCEDURE( ) method, REMOVE-SUPER-PROCEDURE( ) method, RUN SUPER statement, SOURCE-PROCEDURE system handle, SUPER function, SUPER-PROCEDURES attribute, TARGET-PROCEDURE system handle

Temp-table object handle

A handle to a temp-table object. A temp-table object handle corresponds to an underlying Progress temp-table, which can be static or dynamic. A static temp-table is one you define at compile time with the DEFINE TEMP-TABLE statement. A dynamic temp-table is one you create at run time with the CREATE TEMP-TABLE statement.

Syntax

```
temp-table-handle [ :attribute | :method ]
```

temp-table-handle

An item of type WIDGET-HANDLE representing a handle to a temp-table object.

attribute

An attribute of the temp-table object.

method

A method of the temp-table object.

Attributes

AFTER-TABLE attribute	BATCH-SIZE attribute	BEFORE-TABLE attribute
DATA-SOURCE-MODIFIED attribute	DEFAULT-BUFFER-HANDLE attribute	DYNAMIC attribute
ERROR attribute	ERROR-STRING attribute	HANDLE attribute
HAS-RECORDS attribute	LAST-BATCH attribute	MIN-SCHEMA-MARSHAL attribute
NAME attribute	NAMESPACE-PREFIX attribute	NAMESPACE-URI attribute
NO-SCHEMA-MARSHAL attribute	NUM-REFERENCES attribute	ORIGIN-HANDLE attribute
PREPARED attribute	PRIMARY attribute	REJECTED attribute
SCHEMA-MARSHAL attribute	TRACKING-CHANGES attribute	UNDO attribute

Methods

ADD-FIELDS-FROM( ) method	ADD-INDEX-FIELD( ) method
ADD-LIKE-FIELD( ) method	ADD-LIKE-INDEX( ) method
ADD-NEW-FIELD( ) method	ADD-NEW-INDEX( ) method
CLEAR( ) method	COPY-TEMP-TABLE( ) method
CREATE-LIKE( ) method	READ-XML( ) method
READ-XMLSCHEMA( ) method	TEMP-TABLE-PREPARE( ) method
WRITE-XML( ) method	WRITE-XMLSCHEMA( ) method

Example

The following code fragment demonstrates the creation, definition and use of a temp-table object:

```
DEFINE VARIABLE tth AS HANDLE.
DEFINE VARIABLE bh AS HANDLE.
DEFINE VARIABLE qh AS HANDLE.
DEFINE VARIABLE buf-cust-handle AS HANDLE.

/* get db table handle as usual */
buf-cust-handle = BUFFER customer:HANDLE.
/* create an "empty" undefined temp-table */
CREATE TEMP-TABLE tth.
/* give it customer's fields and indexes */
tth:CREATE-LIKE(buf-cust-handle).
/* give it a single extra field */
tth:ADD-NEW-FIELD("f1","integer").
/* no more fields or indexes will be added to custx */
tth:TEMP-TABLE-PREPARE("custx").
/* get the buffer handle for the temp-table */
bh = tth:DEFAULT-BUFFER-HANDLE.
/*populate the table from customer table */
FOR EACH customer:
    bh:BUFFER-CREATE.
    bh:BUFFER-COPY(buf-cust-handle).
END.
/*run a query to access it*/
CREATE QUERY qh.
qh:SET-BUFFERS(bh).
qh:QUERY-PREPARE("for each custx where . . .").
. . .
```

Notes

- The temp-table object has three states, CLEAR, UNPREPARED and PREPARED. The temp-table is in a CLEAR state either when the temp-table is first created or immediately after the CLEAR( ) method is applied. The temp-table is in an UNPREPARED state during the period after the first definitional method has been applied and before the TEMP-TABLE-PREPARE( ) method is applied. The temp-table is in a PREPARED state after the TEMP-TABLE-PREPARE( ) method has been applied.
- The user can discern whether the temp-table is in an UNPREPARED or PREPARED state by using the PREPARED attribute.

See also

Buffer object handle, CREATE TEMP-TABLE statement, DEFINE TEMP-TABLE statement, ProDataSet object handle

THIS-PROCEDURE system handle

A handle to the current procedure object. This object allows you to read and modify the context of the current procedure.

Syntax

```
procedure-handle [ :attribute | :method ]
```

procedure-handle

A handle variable that references a procedure object.

For Web services, this object is instantiated when you execute the RUN ON statement that references a Web service server object.

attribute

An attribute of a procedure handle.

method

Specifies a method of a procedure handle.

Attributes

ADM-DATA attribute	ASYNC-REQUEST-COUNT attribute	CURRENT-WINDOW attribute
DB-REFERENCES attribute	FILE-NAME attribute	INTERNAL-ENTRIES attribute
NAME attribute	NEXT-SIBLING attribute	PERSISTENT attribute
PREV-SIBLING attribute	PRIVATE-DATA attribute	PROXY attribute
PUBLISHED-EVENTS attribute	REMOTE attribute	SERVER attribute
SUPER-PROCEDURES attribute	TRANSACTION attribute	TYPE attribute
UNIQUE-ID attribute		

Methods

ADD-SUPER-PROCEDURE( ) method	GET-SIGNATURE( ) method
REMOVE-SUPER-PROCEDURE( )method	SET-CALLBACK-PROCEDURE( ) method

Examples

The following procedure is designed to run both persistently and non-persistently. It sets up a query on the customer table of the sports database that is selectable by name or balance.

r-thispr.p *(1 of 2)*

```
DEFINE QUERY custq FOR customer.
DEFINE BROWSE custb QUERY custq
    DISPLAY name balance phone WITH 10 DOWN.
DEFINE BUTTON bName LABEL "Query on Name".
DEFINE BUTTON bBalance LABEL "Query on Balance".
DEFINE BUTTON bCancel LABEL "Cancel".

DEFINE FRAME CustFrame custb SKIP
    bName bBalance bCancel.

DEFINE VARIABLE custwin AS WIDGET-HANDLE.

ON CHOOSE OF bName IN FRAME CustFrame DO:
    custwin:TITLE = "Customers by Name".
    OPEN QUERY custq FOR EACH customer BY name.
END.

ON CHOOSE OF bBalance IN FRAME CustFrame DO:
    custwin:TITLE = "Customers by Balance".
    OPEN QUERY custq FOR EACH customer BY balance DESCENDING.
END.
```

r-thispr.p *(2 of 2)*

```
IF THIS-PROCEDURE:PERSISTENT THEN DO:
    THIS-PROCEDURE:PRIVATE-DATA = "Customer Browse".
    CREATE WIDGET-POOL.
END.

CREATE WINDOW custwin
    ASSIGN
        TITLE = "Customer Browser"
        SCROLL-BARS = FALSE
        MAX-HEIGHT-CHARS = FRAME CustFrame:HEIGHT-CHARS
        MAX-WIDTH-CHARS = FRAME CustFrame:WIDTH-CHARS.

THIS-PROCEDURE:CURRENT-WINDOW = custwin.

ENABLE ALL WITH FRAME CustFrame.

IF THIS-PROCEDURE:PERSISTENT THEN DO:
    ON CHOOSE OF bCancel IN FRAME CustFrame DO:
        RUN destroy-query.
    END.
END.
ELSE DO:
    WAIT-FOR CHOOSE OF bCancel IN FRAME CustFrame.
END.

PROCEDURE destroy-query:
    DELETE PROCEDURE THIS-PROCEDURE.
    DELETE WIDGET-POOL.
END.
```

The procedure uses the THIS-PROCEDURE handle to distinguish between persistent and non-persistent instances of execution. When `r-thispr.p` is persistent (THIS-PROCEDURE:PERSISTENT = TRUE), it:

- Sets the PRIVATE-DATA attribute to help identify it to other procedures.
- Creates a private widget pool to maintain its dynamic window for as long as the procedure instance persists.
- Defines a trigger to delete the procedure when it is terminated. Note that the trigger calls the internal procedure destroy-query, which can be executed by other external procedures to delete `r-thispr.p` when it is persistent. This destroy-query routine references the THIS-PROCEDURE handle to delete its persistent parent. It also deletes the widget pool that maintains the dynamic window.

When `r-thispr.p` is non-persistent (THIS-PROCEDURE:PERSISTENT = FALSE), it invokes a WAIT-FOR statement rather than defining a trigger to terminate the procedure. It does not need to create a widget pool or maintain any other persistent context.

Note that because both persistent and non-persistent instances of this procedure use a dynamic window separate from the default window, `r-thispr.p` assigns the window's handle to the procedure's CURRENT-WINDOW attribute. This makes the dynamic window current whether or not the procedure is persistent. However, when the procedure is persistent, the CURRENT-WINDOW attribute keeps the dynamic window current while other procedures execute using different windows. Because the persistent procedure has its own current window, its triggers and internal procedures do not have to reset the current window every time they execute.

Notes

- The attributes supported by THIS-PROCEDURE are supported by any valid procedure handle. You can also define triggers for procedure handles. For more information, see *OpenEdge Development: Progress 4GL Handbook*.
- By determining if the current procedure is persistent, you can decide whether or not to perform certain actions. An action that you might perform during a non-persistent procedure is to execute a WAIT-FOR statement to provide interactive I/O blocking. Actions that you might execute during a persistent procedure include creating a new window to parent all other widgets created in the procedure, or maintaining an unscoped record buffer that lasts as long as the procedure persists.
- To create an instance of a persistent procedure, use the PERSISTENT option of the RUN statement.
- If THIS-PROCEDURE is persistent and the NEXT-SIBLING or PREV-SIBLING attributes are invalid, THIS-PROCEDURE specifies the last or first persistent procedure instance (respectively) in the session persistent procedure chain. To check the validity of these attributes, use the VALID-HANDLE function.
- You can access the handles and attributes of all persistent procedure instances in a session using the FIRST-PROCEDURE or LAST-PROCEDURE attribute of the SESSION handle.
- For information on transaction objects, see *OpenEdge Application Server: Developing AppServer Applications*.

See also RUN statement, SESSION system handle, VALID-HANDLE function

Transaction object handle

Provides access to the current transaction object. This object allows you to control the current transaction context.

Syntax

```
transaction-handle [ :attribute | :method ]
```

transaction-handle

A handle variable whose value you return from the TRANSACTION attribute on a procedure handle.

attribute

An attribute of the transaction handle.

method

A method of the transaction handle.

Attributes

DEFAULT-COMMIT attribute	IS-OPEN attribute	TRANS-INIT-PROCEDURE attribute

Methods

SET-COMMIT( ) method	SET-ROLLBACK( ) method

Notes

- In an AppServer session, if a transaction initiating procedure is active, this handle allows you to control the (automatic) transaction using all of the supported attributes and methods. For more information on automatic transactions, see the TRANSACTION-MODE AUTOMATIC statement reference entry.
- In an OpenEdge client session or in an AppServer session with no active transaction initiating procedure, only the IS-OPEN attribute is available.

- If a transaction initiating procedure is deleted, any open transaction is committed or rolled back according to the value of the DEFAULT-COMMIT attribute.
- The value of this attribute remains the same (references the same transaction context) for the duration of an OpenEdge session. This is true:
  - Whether or not a transaction is opened or closed.
  - In an AppServer session, whether or not a transaction initiating procedure is created or deleted.

See also TRANSACTION-MODE AUTOMATIC statement

WEB-CONTEXT system handle

Provides access to information on the current connection to the Web server.

Note: Applies to SpeedScript programming, not Progress.

Syntax

```
WEB-CONTEXT [ :attribute | :method ]
```

attribute

An attribute of the WEB-CONTEXT handle.

method

A method of the WEB-CONTEXT handle.

Attributes

AUTO-DELETE-XML attribute	CONFIG-NAME( ) attribute	CURRENT-ENVIRONMENT attribute
EXCLUSIVE-ID attribute	FORM-INPUT attribute	FORM-LONG-INPUT attribute
HANDLE attribute	HTML-CHARSET attribute	HTML-END-OF-LINE attribute
HTML-END-OF-PAGE attribute	HTML-FRAME-BEGIN attribute	HTML-FRAME-END attribute
HTML-HEADER-BEGIN attribute	HTML-HEADER-END attribute	HTML-TITLE-BEGIN attribute
HTML-TITLE-END attribute	INSTANTIATING-PROCEDURE attribute	IS-XML attribute
SESSION-END attribute	TYPE attribute	VALIDATE-XML attribute
X-DOCUMENT attribute	XML-SCHEMA-PATH attribute	XML-SUPPRESS-NAMESPACE-PROCESSING attribute

Methods

GET-BINARY-DATA( ) method	GET-CGI-LIST( ) method
GET-CGI-LONG-VALUE( ) method	GET-CGI-VALUE( ) method
GET-CONFIG-VALUE( ) method	INCREMENT-EXCLUSIVE-ID( ) method
URL-DECODE( ) method	URL-ENCODE( ) method

X-document object handle

A handle to an X-document object. You create the handle and assign it to a handle variable with the CREATE X-DOCUMENT statement.

Syntax

```
x-document-handle [ :attribute | :method ]
```

x-document-handle

A handle variable that references an X-document object.

attribute

An attribute of the X-document object.

method

A method of the X-document object.

Attributes

ENCODING attribute	NAME attribute	NAMESPACE-PREFIX attribute
NAMESPACE-URI attribute	NONAMESPACE-SCHEMA-LOCATION attribute	NUM-CHILDREN attribute
PUBLIC-ID attribute	SCHEMA-LOCATION attribute	SCHEMA-PATH attribute
SUBTYPE attribute	SUPPRESS-NAMESPACE-PROCESSING attribute	SYSTEM-ID attribute
TYPE attribute	UNIQUE-ID attribute	

Methods

ADD-SCHEMA-LOCATION( ) method	APPEND-CHILD( ) method
CREATE-NODE( ) method	CREATE-NODE-NAMESPACE( ) method
GET-CHILD( ) method	GET-DOCUMENT-ELEMENT( ) method
IMPORT-NODE( ) method	INITIALIZE-DOCUMENT-TYPE( ) method
INSERT-BEFORE( ) method	LOAD( ) method
REMOVE-CHILD( ) method	REPLACE-CHILD( ) method
SAVE( ) method	

See also

CREATE X-DOCUMENT statement

X-noderef object handle

A handle to a reference to an XML node. The X-noderef object is a 4GL object that is a reference to any arbitrary node in an XML tree except a document node. You create the handle and assign it to a handle variable with the CREATE X-NODEREF statement.

Syntax

```
x-noderef-handle [ :attribute | :method ]
```

x-noderef-handle

A handle variable that references an X-noderef object. You can use this handle as a parameter or return-value for attributes and methods that provide access to the underlying XML node.

attribute

An attribute of the X-noderef object.

method

A method of the X-noderef object.

Attributes

ATTRIBUTE-NAMES attribute	CHILD-NUM attribute	LOCAL-NAME attribute
NAME attribute	NAMESPACE-PREFIX attribute	NAMESPACE-URI attribute
NODE-VALUE attribute	NUM-CHILDREN attribute	OWNER-DOCUMENT attribute
SUBTYPE attribute	TYPE attribute	UNIQUE-ID attribute

Methods

APPEND-CHILD( ) method	CLONE-NODE( ) method
DELETE-NODE( ) method	GET-ATTRIBUTE( ) method
GET-ATTRIBUTE-NODE( ) method	GET-CHILD( ) method
GET-PARENT( ) method	INSERT-BEFORE( ) method
LONGCHAR-TO-NODE-VALUE( ) method	MEMPTR-TO-NODE-VALUE( ) method
NODE-VALUE-TO-LONGCHAR( ) method	NODE-VALUE-TO-MEMPTR( ) method
NORMALIZE( ) method	REMOVE-ATTRIBUTE( ) method
REMOVE-CHILD( ) method	REPLACE-CHILD( ) method
SET-ATTRIBUTE( ) method	SET-ATTRIBUTE-NODE( ) method

See also

CREATE X-NODEREF statement

Attributes and Methods Reference

This section describes each attribute, property, and method that Progress supports. Attributes, properties, and methods are all mechanisms that allow you to monitor and control the behavior of Progress objects, including widget and COM objects. For attributes and methods that apply to SpeedScript, see the "Handle Reference" section on page 1379.

Each Progress widget has a set of attributes and methods. Each COM object has a set of properties and methods. This section describes every widget attribute and method available in Progress, but describes only the COM object properties and methods that directly support the ActiveX control container technology in Progress. All other Automation objects and ActiveX controls that you access from Progress provide their own properties and methods. For more information on these, see the documentation that comes with each COM object.

In this section, names of widget attributes and methods appear in all-uppercase, while names of COM object properties and methods, which follow Visual Basic coding conventions, appear in mixed case.

This section begins by explaining the syntax for widget and COM object references. The basic syntax is similar for both widgets and COM objects. However, it has been extended for COM objects to support the unique features of Automation objects and ActiveX controls.

Note: In character interfaces, all attributes and methods that reference pixels (for example, the HEIGHT-PIXELS attribute and the GET-TEXT-HEIGHT-PIXELS method) use a system default pixel value for the equivalent value in characters.

Referencing widget attributes and methods

A **widget attribute** is a value that defines the visible, functional, and other characteristics of a widget or procedure. System handles also have attributes that describe and control certain widget or system states. Attributes can be readable, writeable, or both. Readable means that your code can assign the value of the attribute to a variable or reference its value in an expression. Writeable means that your code can change the value of an attribute and thereby change the associated characteristic of the widget. Whether or not an attribute is readable or writeable depends on a number of factors (for example, the widget type, system handle type, widget realization, etc.).

A **widget method** is a specialized function associated with a widget that performs an action on the widget or alters the behavior of the widget. Some system handles also have methods that affect certain widget and system behaviors. All methods return a value and some methods require parameters. The return value usually is a logical value specifying whether or not the execution of the method was successful; however, some methods return other types of information.

Widget attribute references

To reference an attribute, use the following syntax:

```
{ widget-name-reference | handle }
  :attribute-name
    [ IN container-widget-name ]
```

A *widget-name-reference* is a name reference to a static widget. A *handle* is a widget handle, procedure handle, or system handle reference. A *container-widget-name* is a name reference to a static container widget for *widget-name-reference* or to a browse widget. You need it only if *widget-name-reference* is ambiguous. For more information on attribute references, see the chapter on widgets and handles in *OpenEdge Development: Progress 4GL Handbook*.

The attributes in this section are listed alphabetically by name. Each attribute entry defines the data type of the attribute and provides information about read/write status of the attribute.

To read an attribute value, assign the attribute reference to a field or variable of a compatible data type or include the attribute reference in an expression. To write an attribute value, assign the value to an attribute reference.

The following example repositions a selection list (Select-1) to another row in its frame (SelectFrame):

```
Select-1:ROW IN FRAME SelectFrame = Select-1:ROW + 2.
```

Chained widget attributes

The Chained widget attribute lets you supply multiple attributes for a widget. When you use this syntax, the middle attributes and methods must have a data type of HANDLE. The widget-name-reference specifies a widget object. For example, SELF, CURRENT-WINDOW, BUFFER customer:HANDLE, or Query q:handle. The *handle* can be any variable of type HANDLE, and the *attribute-name* can be any attribute for the widget or handle immediately preceding it.

The Chained widget attribute has the following syntax:

```
{widget-name-reference | handle}: <attribute-name> [:attribute-name]...
```

The following shows an example of Chained widget attributes:

```
DEFINE VARIABLE hBuff as HANDLE.
CREATE BUFFER hBuff  FOR TABLE "customer".
MESSAGE hBuff:BUFFER-FIELD(3):NAME.
```

See also Widget phrase, WIDGET-HANDLE function.

Widget method references

To reference a method, use the following syntax:

```
{ widget-name-reference | handle}
  :method-name ( [ parameter-list ] )
   [ IN container-widget-name ]
```

A *widget-name-reference* is a name reference to a static widget. A *handle* is a widget handle, procedure handle, or system handle reference. A *container-widget-name* is a name reference to a static container widget for the *widget-name-reference* or a browse widget. You need it only if *widget-name-reference* is ambiguous. For more information on method references, see the chapter on widgets and handles in *OpenEdge Development: Progress 4GL Handbook*.

To execute a method, you can assign the return value to a variable, reference the method in an expression, or invoke the method as a statement, ignoring the return value. The following example executes the ADD-FIRST( ) method for a selection list (Select-1) in two different ways-assigning the return value to a logical variable (methRtn) and invoking it directly:

```
methRtn = Select-1:ADD-FIRST("BLUE").
Select-1:ADD-FIRST("GREEN").
```

The methods in this section are listed alphabetically by name. Each method entry defines the data type of the return value and describes any required parameters.

Widget color, font, and measurement values

Some entries in this section refer to color and font values (for example, BGCOLOR and FONT). These values are color and font numbers established for your system. Some entries in this section also refer to character units, a Progress unit of measure for specifying portable widget sizes and positions in graphical environments.

For more information on color and font values, or character units and their relationship to pixels, see *OpenEdge Development: Programming Interfaces*.

Note: When you assign a decimal value to an attribute representing a measurement in character units, Progress automatically rounds the assigned value to the nearest decimal value that corresponds to whole pixel units.

Referencing COM object properties and methods

A **COM object property** is a value that defines the visible, functional, and other characteristics of a COM object (ActiveX Automation object or ActiveX control). An ActiveX control property is classified as a design-time or run-time property depending on when you can change it. A design-time property can be changed using the Properties Window of the AppBuilder. A run-time property can be changed from the 4GL at run time. Generally, you can read both design-time and run-time properties at run time. In all other respects, COM object properties are functionally analogous to widget attributes.

A **COM object method** is a specialized function associated with a COM object that performs an action on the COM object or alters the behavior of the COM object. COM object methods may or may not return a value and may or may not require parameters. A return value may be a component handle to another COM object; however, many methods return other types of information or no information at all. Like widget methods, you execute COM object methods by direct invocation as statements rather than by invocation as part of an expression. In all other respects, COM object methods are functionally analogous to widget methods.

The basic syntax for referencing COM object properties and methods from Progress is similar to widget attribute and method references. The main differences include:

- You might have to specify the parameters of COM object methods with more type information, depending on the methods and how the COM objects are implemented.
- All COM objects are dynamic objects, so you never qualify a COM object reference by a static container reference (such as a static frame or menu widget).

COM object references

Because a chain of COM object property and method references ultimately resolve to a single property or method reference, the syntax diagrams that describe COM object references begin with the three basic types of statements that reference properties and methods:

- Property write
- Property read
- Method call

Property write

```
Com-Handle-Var :COMProperty [ AS Datatype ] = expression
  [ NO-ERROR ]
```

Property read

```
[ { field | COMProperty } = ]
  Com-Handle-Var :COMProperty [ NO-ERROR ]
```

Method call

```
[ { field | COMProperty } = | NO-RETURN-VALUE ]
  Com-Handle-Var :COMMethod [ NO-ERROR ]
```

Com-Handle-Var is any COM-HANDLE variable set to the handle of an instantiated COM object; *field* is any Progress variable, database field, or widget attribute reference of a compatible data type; *expression* is any combination of 4GL elements that results in a single value.

Note: You can invoke both a property read and a method call as part of an expression in another statement (such as in a DISPLAY statement). You can also directly invoke both property reads and method calls as statements in themselves. However, direct invocation is meaningful only for method calls.

The following syntax boxes describe the remaining components of these basic statements:

COMProperty

```
[ { COMProperty | COMMethod } : ] ...
  Property-Name [ ( index [ , index ] ... ) ]
```

Property-Name is the name of the referenced property. The optional multi-level *index* is an integer expression as required by the property. You must not follow the colon separator by a space.

COMMethod

```
[ { COMProperty | COMMethod } : ] ...
  Method-Name ( [ COMparm [ , COMparm ] ... ] )
```

Method-Name is the effective name of the referenced method. *COMparm* is a parameter as required by the method. You must not follow the colon separator by a space.

COMParm

```
{ [ OUTPUT | INPUT-OUTPUT ] expression [ AS Datatype ]
    [ BY-POINTER | BY-VARIANT-POINTER ]
  | null-parm
}
```

A *null-parm* is any amount of white space.

Note: There is currently no support for named parameters, for example: *Method-Name*(Color="GREEN", Shape="SQUARE")*Datatype*

Data type

```
SHORT | FLOAT | CURRENCY | UNSIGNED-BYTE | ERROR-CODE | IUNKNOWN
```

The requirements for using the OUTPUT, INPUT-OUTPUT, BY-VARIANT-POINTER, BY-POINTER, and AS *Datatype* options depend on the COM object method or property, the implementation of the COM object, and how you plan to use the parameter or property in your application. In many cases, *expression* is all that you require for a property write or method parameter. For more information and examples of COM object references, see the chapter on ActiveX Automation and the chapter on control container support in *OpenEdge Development: Programming Interfaces*.

ACCELERATOR attribute

The key label of the keyboard accelerator for the menu item.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: MENU-ITEM widget

ACCEPT-CHANGES( ) method

Accepts changes to the data in one temp-table or all temp-tables in a ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle

Syntax

```
handle:ACCEPT-CHANGES( )
```

handle

A handle to the temp-table buffer or the ProDataSet object.

When you accept changes on a ProDataSet object handle, Progress makes the rows in all after-image tables the current version of those rows, and empties the before-image tables. When you accept changes for a Buffer object handle, Progress makes the rows in the after-image table the current version of those rows, and empties the before-image table. In either case, Progress sets the BEFORE-ROWID attribute of every row in the after-image tables to the Unknown value (?), and the ROW-STATE of every row in the after-image tables to ROW-UNMODIFIED (0).

ACCEPT-ROW-CHANGES( ) method

Accepts changes to the data in one row of a ProDataSet temp-table.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
handle:ACCEPT-ROW-CHANGES( )
```

handle

A handle to a before-image temp-table buffer.

When you accept changes for a temp-table row, Progress makes the row in the after-image table the current version of the row, and then removes the before-image table row. Progress also sets the BEFORE-ROWID attribute of the row in the after-image table to the Unknown value (?), and the ROW-STATE of the row in the after-image table to ROW-UNMODIFIED (0).

ACTIVE attribute

Indicates whether an individual data-relation between two ProDataSet object buffers is active or inactive. Set to TRUE to activate an individual data-relation. Set to FALSE to deactivate a data-relation.

Alternatively, you can activate or deactivate all data-relations in a ProDataSet object by setting the RELATIONS-ACTIVE attribute on the ProDataSet object handle. All data-relations in a ProDataSet object are active by default.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-relation object handle

Deactivate an individual data-relation in a ProDataSet object when you want a FILL operation to load data into a ProDataSet member buffer using the individual buffer's query. Likewise, you can reactivate a data-relation in a ProDataSet object after completing a FILL operation to use the data-relation for traversing the data after the data is loaded.

When Progress encounters an inactive relation (or the last child buffer in the relation tree), during a FILL operation on a ProDataSet buffer object handle, Progress does not fill the child buffers of that relation. When Progress encounters an inactive relation during a FILL operation on a ProDataSet object handle, it treats the first child buffer of the inactive relation as a top-level table (including all rows from its data source) and fills each child buffer based on the data relation's query. If you do not want Progress to treat the first child buffer of the inactive relation as a top-level table, set the FILL-MODE of that buffer to NO-FILL. Progress does not fill any of the child buffers.

If Progress encounters an inactive relation while navigating a ProDataSet object, it does not prepare or open a dynamic query for the child table, even if there is a browse associated with the relation's query. If you want to access the child temp-table, you must do so through a separate query, a FOR EACH statement, or some other standard Progress construct in your application code.

When you reactivate data-relations, Progress does not automatically resynchronize the hierarchy of queries on buffers below the newly active relation. If you want to resynchronize the related buffers, use the SYNCHRONIZE( ) method on the parent buffer.

ACTOR attribute

Returns the value of the actor attribute for the SOAP-header-entryref object as a URL. Identifies the recipient of a header element.

Data type: CHARACTER

Access: Readable

Applies to: SOAP-header-entryref object handle

If the SOAP-header-entryref object does not contain an actor attribute, this attribute returns the empty string.

ADD-BUFFER( ) method

Adds one new buffer to a query object or dynamic ProDataSet object, without affecting the other buffers, if any.

Use the SET-BUFFERS( ) method to remove all prior buffers and set all buffers for the object at the same time.

Return type: LOGICAL

Applies to: ProDataSet object handle, Query object handle

Syntax

```
ADD-BUFFER ( buffer )
```

buffer

A handle to a buffer, or a CHARACTER expression that evaluates to the name of a buffer that Progress searches for at run time.

Note: The maximum number of buffers per query is 18.

The following is an example:

```
my-query-handle:ADD-BUFFER(BUFFER customer:handle).
```

ADD-CALC-COLUMN( ) method (Windows only)

Creates a browse column from the specified properties and returns the widget-handle of the new column. This method can be used only after the browse's query attribute has been set.

Return type: WIDGET-HANDLE

Applies to: BROWSE widget

Syntax

```
ADD-CALC-COLUMN( datatype-exp , format-exp , initial-value-exp , label-exp
[ , pos ] )
```

datatype-exp

Character expression specifying the data type ("CHAR", "DATE", "DECIMAL", "INTEGER" or "LOGICAL")

format-exp

Character expression specifying the column's format.

initial-value-exp

Character expression specifying the initial value. This may be a null string.

label-exp

Character expression specifying the column's label.

pos

The optional integer value position of the browse column. If *pos* = 2, the column is the second column. If the position is not specified or the position is invalid, the new column is added at the end of the columns.

The following is an example of adding a column in the browse's fifth position using this method:

```
CalcHnd5 = BrwsHndl:ADD-CALC-COLUMN("char", "AAA-99999", "ORD-37854",
"OrdNum", 5).
```

Notes

- The ADD-CALC-COLUMN( ) method may be used on a static browse as well as on a dynamic browse.
- If the browse is already displayed, the REFRESH( ) method should be applied to the browse after columns are added using ADD-CALC-COLUMN( ). This will initially populate the viewport for the calculated column. The ROW-DISPLAY trigger would normally populate the column, but when ADD-CALC-COLUMN is being executed, the 4GL calc-column handle is not yet set and, thus, cannot initially populate it.

ADD-COLUMNS-FROM( ) method (Windows only)

Creates a browse column for each field found in the specified buffer or table. If a field is found that already has a corresponding browse-column created, it is ignored. This method can be used only after the browse's query attribute has been set.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
ADD-COLUMNS-FROM( buffer-hndl | table-name-exp [ , except-list ] )
```

buffer-hndl

The widget handle of a buffer associated with the browse's query.

table-name-exp

The name of a table associated with the browse's query.

except-list

An expression that evaluates to a comma-separated list of field names to be excluded from the browse. No extra spaces should be included between names.

The following is an example of adding browse columns from the Invoice table, excluding the fields, Amount and Total-Paid:

```
DEFINE VARIABLE ExcList AS CHARACTER INITIAL "Amount,Total-Paid".
...
AddCol = BrwsHndl:ADD-COLUMNS-FROM("invoice", ExcList).
```

Notes

- The ADD-COLUMNS-FROM( ) method may be used on a static browse as well as on a dynamic browse. When used on a static browse, the browse will become a NO-ASSIGN browse (you must make the database updates.)
- A dynamic browse column's validation expression is restricted. It may not contain a CAN-FIND function. To reference the field, the FRAME-VALUE function must be used.

ADD-EVENTS-PROCEDURE( ) method (Windows only; Graphical interfaces only)

Adds an external procedure to the list that Progress searches for event procedures to handle ActiveX control events.

Return type: LOGICAL

Applies to: CONTROL-FRAME widget

Syntax

```
ADD-EVENTS-PROCEDURE ( procedure-handle )
```

procedure-handle

A handle to a persistent procedure or an otherwise active procedure on the call stack.

By default, Progress searches the external procedure that created the current control-frame for the event procedure to handle an ActiveX control event. This method allows you to specify alternative procedure (.p and .w) files to search for the event handler.

When Progress receives an ActiveX event, it searches for the event handler in order of the most recent procedure added to the search list and ends the search with the external procedure that created the control-frame. You can override an existing procedure by adding a different one to the search list. Progress always uses the event handler in the most recently added procedure.

If the method succeeds in adding the procedure to the list, it returns TRUE. Otherwise, it returns FALSE.

ADD-FIELDS-FROM( ) method

Copies the field definitions from the specified source table to a temp-table. It is intended for use when a temp-table represents a join. If it finds fields that are already in the temp-table, it ignores them.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-FIELDS-FROM( { source-table-hndl-exp | source-table-name-exp }
   [ , except-list-exp ] )
```

source-table-hndl-exp

An expression that evaluates to a table handle from which to copy the field definitions.

source-table-name-exp

An expression that evaluates to a table name from which to copy the field definitions.

except-list-exp

A character expression that evaluates to a comma-separated list of field names to be excluded from the new table definition.

This method does not create any indexes. Either indexes must be added specifically through one of the ADD-INDEX methods, or a default index is created.

The following example fragment creates a join temp-table from the customer and order tables:

```
DEFINE tth AS HANDLE.
CREATE TEMP-TABLE tth.
tth:ADD-FIELDS-FROM("customer").
tth:ADD-FIELDS-FROM("order").
tth:TEMP-TABLE-PREPARE("cust-ord").
. . .
```

The following fragment creates a temp-table from the customer table except for the sales-rep field:

```
tth:ADD-FIELDS-FROM("customer","sales-rep").
```

Note: There is a limit to the number of fields that can be accommodated in a temp-table object. The limit depends on how large the field information (initial value, validate information, help messages, etc.) is, but you should plan on a limit of 500 fields.

ADD-FIRST( ) method

Adds one or more items to the top of a combo box or selection list.

Return type: LOGICAL

Applies to: COMBO-BOX widget, SELECTION-LIST widget

Syntax

```
ADD-FIRST ( { item-list | label , value } )
```

item-list

A character-string expression that represents one or more items, delimiter-separated.

label

A character-string expression that represents the label of a label-value pair.

value

The value Progress assigns to the field or variable if the user selects the corresponding label.

The delimiter is the value of the DELIMITER attribute, which defaults to comma. If the SORT attribute is TRUE, Progress sorts new items by label before adding them to the widget. If the operation is successful, ADD-FIRST returns TRUE.

Notes

- If the widget's entries consist of single items, use *item-list*. If the widget's entries consist of label-value pairs, use *label* and *value*.
- If the widget's entries consist of single items, each call to ADD-FIRST can add multiple entries. If the widget's entries consist of label-value pairs, each call to ADD-FIRST can add one entry.

Examples

The following examples modify widgets whose entries consist of single items:

```
return-code = my-widget-hdl:ADD-FIRST("Seoul").
```

```
return-code = my-widget-hdl:ADD-FIRST("Bogota, Seoul, Los Angeles").
```

The following example modifies a combo-box widget of type INTEGER whose entries consist of label-value pairs:

```
return-code = my-widget-hdl:ADD-FIRST("Bogota", 15).
```

ADD-HEADER-ENTRY( ) method

Creates a SOAP-header-entryref object, attaches it to the SOAP-header object, and returns the handle to the new header entry.

Return type: LOGICAL

Applies to: SOAP-header object handle

Syntax

```
ADD-HEADER-ENTRY( header-entryref )
```

header-entryref

The handle to the new SOAP-header-entryref object.

Following is an example of adding a SOAP-header-entryref object to a SOAP-header object:

```
CREATE SOAP-HEADER hSOAPHeader.
CREATE SOAP-HEADER-ENTRYREF hshEntry.
hSOAPHeader:ADD-HEADER-ENTRY (hshEntry).
```

ADD-INDEX-FIELD( ) method

Adds the specified field to the specified index of a temp-table. It requires the named index to be added first.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-INDEX-FIELD( index-name-exp , field-name-exp [ , mode-exp ] )
```

index-name-exp

A character expression that evaluates to the name of the index to which the field is being added.

field-name-exp

A character expression that evaluates to the name of the field to add to the index.

mode-exp

An expression that evaluates to desc if it is descending or asc if it is ascending. The default is asc.

The following example fragment adds to a temp-table a new unique primary index field with two components, the first ascending, the second descending:

```
tth:ADD-FIELDS-FROM("customer","sales-rep").
tth:ADD-NEW-INDEX("abidx",true,true).
tth:ADD-INDEX-FIELD("abidx","abfield1").
tth:ADD-INDEX-FIELD("abidx","abfield2","desc").
...
```

ADD-LAST( ) method

Adds one or more items to the bottom of a combo box, radio set, or selection list.

Return type: LOGICAL

Applies to: COMBO-BOX widget, RADIO-SET widget, SELECTION-LIST widget

Combo-box and selection-list syntax

```
ADD-LAST ( { item-list | label , value } )
```

item-list

A character-string expression that represents one or more items, delimiter-separated.

label

A character-string expression that represents the label of a label-value pair.

value

The value Progress assigns to the field or variable if the user selects the corresponding label.

Note: If the widget's entries consist of single items, use *item-list*. If the widget's entries consist of label-value pairs, use *label* and *value*.

For combo boxes and selection lists, the delimiter is the value of the DELIMITER attribute, which is comma by default. Also, if the SORT attribute is TRUE, ADD-LAST sorts the new items by label before adding them to the widget.

Radio-set syntax

```
ADD-LAST ( label , value )
```

label

A character-string expression that represents the label of a label-value pair.

value

An INTEGER expression that represents the value of a label-value pair. When the radio set appears, if the user selects *label*, Progress assigns *value* to the corresponding field or variable.

For radio sets, if the AUTO-RESIZE attribute is TRUE; the size of the radio set changes. Otherwise, the radio set is clipped.

For all applicable widgets, if the operation is successful, ADD-LAST returns TRUE.

Note: If the widget's entries consist of single items, each call to ADD-LAST can add multiple entries. If the widget's entries consist of label-value pairs, each call to ADD-LAST can add one entry.

The following examples modify widgets whose entries consist of single items:

```
return-code = my-combo-box-hdl:ADD-LAST("Seoul").
```

```
return-code = my-sel-list-hdl:ADD-LAST("Bogota, Seoul, Los Angeles").
```

The following example modifies a combo-box widget of type INTEGER whose entries consist of label-value pairs:

```
return-code = my-widget-hdl:ADD-LAST("Bogota", 15).
```

ADD-LIKE-COLUMN( ) method (Windows only)

Creates a browse column from the specified field and returns its widget handle. This method can be used only after the browse's query attribute has been set.

Return type: WIDGET-HANDLE

Applies to: BROWSE widget

Syntax

```
ADD-LIKE-COLUMN( field-name-exp | buffer-field-hndl [ , pos ] )
```

field-name-exp

The name of a field in one of the buffers associated with the browse's query. If the query is a join, the name must be qualified with the database name.

buffer-field-hndl

The widget-handle of a buffer-field from a buffer associated with the browse's query.

pos

The optional integer value position of the browse column. If *pos* = 2, the column is the second column. If the position is not specified or the position is invalid, the new column is added at the end of the columns.

The following is an example of adding the customer number field to the browse:

```
ColHdl = BrwsHndl:ADD-LIKE-COLUMN("customer.cust-num").
```

Notes

- The ADD-LIKE-COLUMN( ) method may be used on a static browse as well as on a dynamic browse. When used on a static browse, the browse will become a NO-ASSIGN browse (you must make the database updates.)
- A dynamic browse column's validation expression is restricted. It may not contain a CAN-FIND function. To reference the field, the FRAME-VALUE function must be used.

ADD-LIKE-FIELD( ) method

Adds a field, like the specified source field, to the temp-table.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-LIKE-FIELD( field-name-exp ,
      source-buffer-field-hndl-exp | source-db-field-name-exp )
```

field-name-exp

A character expression that evaluates to the name of the field to be created in the temp-table.

source-buffer-field-hndl-exp

A character expression that evaluates to a buffer-field handle from which to copy the field.

source-db-field-name-exp

A character expression that evaluates to a database field name from which to copy the field. The table name must be qualified with the database name.

The following example fragments add a field to a temp-table, the first from a named source and the second from a buffer-field handle source:

```
tth:ADD-LIKE-FIELD("ordno","order.order-num").
```

```
tth:ADD-LIKE-FIELD(bfh:name,bfh).
```

Note: There is a limit to the number of fields that can be accommodated in a temp-table object. The limit depends on how large the field information (initial value, validate information, help messages, etc.) is, but you should plan on a limit of 500 fields.

ADD-LIKE-INDEX( ) method

Adds an index, like the specified source index, to the temp-table.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-LIKE-INDEX( index-name-exp , source-index-name-exp
    { , source-buffer-hndl-exp | source-db-table-name-exp } )
```

index-name-exp

A character expression that evaluates to the name of the index to which the source index is being copied.

source-index-name-exp

A character expression that evaluates to the name of the index in the source table that is being copied to the temp-table.

source-buffer-hndl-exp

A character expression that evaluates to a buffer handle from which to copy the index.

source-db-table-name-exp

A character expression that evaluates to a database table name from which to copy the index.

The following example fragment adds a new index to a temp-table like the name index in the customer table:

```
tth:ADD-LIKE-INDEX("abidx","name","customer").
```

ADD-NEW-FIELD( ) method

Adds a field with the specified properties to the temp-table. Additional properties can be manipulated by creating a buffer-field object for this field.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-NEW-FIELD( field-name-exp , datatype-exp [ , extent-exp [ , format-exp
[ , initial-exp [ , label-exp [ , column-label-exp ] ] ] ] ] )
```

field-name-exp

A character expression that evaluates to the name of the field to be created in the temp-table.

datatype-exp

A character expression that evaluates to the data type of the specified field.

extent-exp

An integer expression specifying the extent of an array. If *extent-exp* is 0, 1 or the Unknown value (?), it is ignored.

format-exp

A character expression that evaluates to the data format for the defined data type. If *format-exp* is "" or the Unknown value (?), it is ignored and the default format of the specified data type is used.

initial-exp

An expression that evaluates to the initial value of the defined field. *initial-exp* can be any compatible data type, but is usually character. If *initial-exp* is not entered, the default for the data type is used.

label-exp

An optional character expression that evaluates to the label of the defined field. If you do not specify a value, or you pass the Unknown value (?), *label-exp* defaults to the value of the *field-name-exp* parameter.

column-label-exp

An optional character expression that evaluates to the label of the column associated with the defined field. If you do not specify a value, or you pass the Unknown value (?), *column-label-exp* defaults to the value of the *label-exp* parameter (or the *field-name-exp* parameter, if the *label-exp* parameter is not specified).

The following example fragment adds a new character field called “abfield” which is initialized to “abc” to a temp-table:

```
tth:ADD-NEW-FIELD("abfield","char",0,,"abc").
```

Note: There is a limit to the number of fields that can be accommodated in a temp-table object. The limit depends on how large the field information (initial value, validate information, help messages, etc.) is, but you should plan on a limit of 500 fields.

ADD-NEW-INDEX( ) method

Adds a new empty index with the specified name to the temp-table. Index components must be added with the ADD-INDEX-FIELD( ) method.

This method cannot be called after TEMP-TABLE-PREPARE( ) has been called unless CLEAR( ) is called first.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
ADD-NEW-INDEX( index-name-exp [ , unique-exp [ , primary-exp
    [ , wordix-exp ] ] ] )
```

index-name-exp

A character expression that evaluates to the name of the created index.

unique-exp

A logical expression that evaluates to TRUE if this index is unique.

primary-exp

A logical expression that evaluates to TRUE if this is the primary index.

wordix-exp

A logical expression that evaluates to TRUE if this is a word index.

The following example fragment adds to a temp-table a new unique primary index field with two components, the first ascending, the second descending:

```
tth:ADD-FIELDS-FROM("customer","sales-rep").
tth:ADD-NEW-INDEX("abidx",true,true).
tth:ADD-INDEX-FIELD("abidx","abfield1").
tth:ADD-INDEX-FIELD("abidx","abfield2","desc").
...
```

ADD-RELATION( ) method

Adds a data-relation object for a pair of parent and child buffers to a dynamic ProDataSet object.

Return type: HANDLE

Applies to: ProDataSet object handle

Syntax

```
ADD-RELATION ( parent-buffer-handle, child-buffer-handle,
   [ pairs-list [ , reposition-mode [, nested ] ] ])
```

parent-buffer-handle

A handle to the parent buffer in the data-relation object.

child-buffer-handle

A handle to the child buffer in the data-relation object.

pairs-list

An expression that evaluates to a comma-separated list of parent-field, child-field pairs describing the relationship between parent and child buffers in the data-relation object using the following syntax:

```
parent-field1, child-field1 [, parent-fieldn, child-fieldn ] ...)
```

The first field in the pair is from the parent buffer, the second field is from the child buffer. When filling the ProDataSet object, Progress retrieves data for the child buffer based on an equality match between all pairs of fields unless the Data-Relation is deactivated or there is an explicit query definition for the data source of the child buffer.

You can define a query for the data source of the child buffer, or supply custom logic in response to FILL events that take over complete responsibility for filling one level of the ProDataSet object. In these cases, the *pairs-list* is not used.

reposition-mode

The reposition mode of the relation between the parent and child temp-tables. If TRUE, the relation mode is REPOSITION. If FALSE, the relation mode is SELECTION. The default value is FALSE.

When the relation mode is SELECTION, the method fills the child temp-table of the data-relation object with all records related to the current parent. When the relation mode is REPOSITION, the relation is effectively ignored during a FILL, and the child of the relation is treated as if it were a top-level buffer.

When navigating a filled ProDataSet object with a SELECTION relation, related data is filtered as it is browsed. This means the child query of the relation is filtered to make available only children of the current parent, and the query is re-opened each time the parent table is repositioned. When navigating a filled ProDataSet object with a REPOSITION relation, the child table query is always set to match all the rows in the child table, and is not re-opened when the parent changes. Only the buffer for the child is repositioned to the matching child for the current parent.

nested

A LOGICAL expression where TRUE directs Progress to nest child rows of ProDataSet buffers within their parent rows when writing the XML representation of data. This also causes the XML Schema definitions for the related temp-tables to be nested. When FALSE, all child rows are written after all parent rows are written. The default value is FALSE.

ADD-SCHEMA-LOCATION( ) method

An XML Schema file location is specified by providing a pair of values: a namespace and a physical location. This method allows you to specify that value pair. The XML Schema file is used by an X-document or SAX-reader object to validate XML content.

Return type: LOGICAL

Applies to: X-document object handle, SAX-reader object handle

Syntax

```
ADD-SCHEMA-LOCATION( targetNamespace, location )
```

targetNamespace

A CHARACTER expression evaluating to the target namespace of the schema, or an empty string ("") or the Unknown value (?) if the location doesn't contain a namespace.

location

A CHARACTER expression evaluating to the location of the XML Schema file.

Provides the location of XML Schema file for the parser by specifying the namespace and physical location of the XML Schema file.

You can call this method more than once to create a list of schema locations.

Note that namespace and XML Schema file locations specified programatically with this method take precedence over namespaces or schemas declared in XML documents or imported elements.

This method and the SCHEMA-LOCATION attribute are both used for setting the XML Schema file location. This method is added for convenience.

ADD-SOURCE-BUFFER( ) method

Adds a database buffer to a dynamic Data-source object at runtime.

Return type: LOGICAL

Applies to: Data-source object handle

Syntax

```
ADD-SOURCE-BUFFER ( buffer-handle, key-fields )
```

buffer-handle

A handle to the database buffer you are adding.

key-fields

A character expression that evaluates to a comma-separated list of one or more database table fields that constitute a unique key that Progress can use to find a record in the table.

ADD-SUPER-PROCEDURE( ) method

Associates a super procedure file with a procedure file or with the current OpenEdge session. When a procedure file invokes an internal procedure or a user-defined function, Progress searches for it, among other places, in the super procedures (if any) of the procedure file and of the current OpenEdge session. The procedure-search option determines which procedures are searched.

For more information on the rules that Progress uses to search for internal procedures and user-defined functions, see the "Search rules" section on page 1531. For information on super procedures, see *OpenEdge Development: Progress 4GL Handbook*. For a sample program that uses the ADD-SUPER-PROCEDURE method, see the reference entry for the RUN SUPER statement in this book.

Returns FALSE for a Web service procedure.

Return type: LOGICAL

Applies to: SESSION system handle, THIS-PROCEDURE system handle and all procedure handles

Syntax

```
ADD-SUPER-PROCEDURE ( super-proc-hdl [ , proc-search ] )
```

super-proc-handle

The handle of a running persistent procedure that you want to make a super procedure of the local procedure or of the current OpenEdge session.

ADD-SUPER-PROCEDURE returns FALSE if *super-proc-hdl* is not a valid handle, or if Progress detects that the method was not successful. Otherwise, the method returns TRUE.

proc-search

Optional expression that determines which super procedures are searched when super-proc-hdl invokes RUN SUPER or the SUPER function. Valid values are SEARCH-SELF (or 1) or SEARCH-TARGET (or 2). The default, if there is no entry, is SEARCH-SELF. The search commences in the super procedure stack of *super-proc-hdl*.

Consider the following:

- SEARCH-SELF starts searching in the procedure file that initiated the current internal procedure or user-defined function.
- SEARCH-TARGET starts searching the super procedures of the procedure file that originally invoked the current internal procedure or user-defined function (the procedure with the original RUN statement). If the procedure was RUN . . . IN *procedure-handle*, SEARCH-TARGET searches the super procedures of *procedure-handle*.
- A given *super-proc-hdl* can be added as either SEARCH-TARGET or SEARCH-SELF, but cannot be added as both. If *proc-search* is set for a *super-proc-hdl*, then any attempt to change its value generates a run-time warning, but the ADD-SUPER-PROCEDURE( ) method succeeds. The warning message "Changing proc-search-string for procedure <.p-name> from <string> to <string>" is presented to indicate that the application is using an instance of a given super procedure in an inconsistent manner. This warning message can be suppressed by using the SESSION:SUPPRESS-WARNINGS attribute. In addition, the warning message can be avoided by creating two instances of *super-proc-hdl*, one identified as SEARCH-TARGET and the other identified as SEARCH-SELF.

Associating a super procedure with a procedure

The following example associates a super procedure with the current procedure:

```
THIS-PROCEDURE:ADD-SUPER-PROCEDURE(my-super-proc-hdl,SEARCH-SELF).
```

The following example:

- Associates a super procedure with a procedure that the current procedure is working for.
- Requests that the super procedure stack associated with local-proc-hdl be searched rather than the stack associated with my-super-proc-hdl when RUN SUPER is invoked in *super-proc-hdl*.

```
local-proc-hdl:ADD-SUPER-PROCEDURE(my-super-proc-hdl,SEARCH-TARGET).
```

The procedure to which you add a super procedure is called the *local procedure* of the super procedure.

Associating a super procedure with the current OpenEdge session

The following example associates a super procedure with the current OpenEdge session:

```
SESSION:ADD-SUPER-PROCEDURE(my-super-proc-hdl).
```

When you do this, Progress automatically associates the super procedure with all the session's procedures—persistent and nonpersistent—without your having to change their code in any way. This technique lets you replace occurrences of the following:

```
THIS-PROCEDURE:ADD-SUPER-PROCEDURE(super-proc-hdl).
```

in individual procedures with a single occurrence of the following:

```
SESSION:ADD-SUPER-PROCEDURE(super-proc-hdl).
```

Super procedure stacking

You can associate multiple super procedures with a single local procedure or with the current OpenEdge session. When you do this, Progress stores (and later on, searches) the corresponding procedure handles in last in first out (LIFO) order—the handle of the most recently added super procedure first, the handle of the next most recently added super procedure second, etc.

A collection of super procedure handles associated with a local procedure or with the current OpenEdge session is called a super procedure **stack**. The handle of the most recently added super procedure occupies the **top** of the stack.

If you add a super procedure that is already in the stack, Progress removes the previous occurrence of the super procedure handle from the stack and adds the new occurrence to the top of the stack—all without reporting an error.

Super procedure chaining

You can add a super procedure to a super procedure. For example, imagine the following scenario:

1. A, B, and C are procedure files running persistently.
2. B is a super procedure of A.
3. C is a super procedure of B.

B is a super procedure (of A) and has a super procedure (C).

When you add a super procedure to a super procedure, the result is a super procedure **chain**, each **link** of which consists of two elements: a local procedure and its super procedure. When Progress searches a super procedure chain, it does not proceed to the next link unless the current link's super procedure element explicitly invokes its super version (by using the RUN SUPER statement or the SUPER function).

For example, imagine the following scenario:

1. A, B, and C, and X are procedure files running persistently.
2. add-record is an internal procedure different versions of which reside in A, B, and C.
3. B is a super procedure of A.
4. C is a super procedure of B.
5. X says RUN add-record IN A.

The following events occur:

1. Progress searches A for add-record and runs it if found.
2. If and only if A's add-record exists and says RUN SUPER, Progress searches B for add-record and runs it if found.

   Note: If A does not contain add-record, the following events occur: If B contains add-record, Progress runs it. If B does not contain add-record, Progress does not search for add-record in C.

3. If and only if B's add-record exists and says RUN SUPER, Progress searches C for add-record and runs it if found.

In this way, Progress avoids excessive and possibly circular searching.

Search rules

Progress searches for internal procedures and user-defined functions depending on how the internal procedure or user-defined function is invoked. The search rules illustrated in the first three cases assume that all the super procedures were added with no *proc-search* value or with a *proc-search* value of SEARCH-SELF. The fourth case illustrates the search process when a super procedure is added with a *proc-search* value of SEARCH-TARGET:

Case 1: When Progress encounters a statement like the following:

```
RUN add-record('customer').
```

Progress searches for add-record as follows:

1. As an internal procedure in the local procedure.
2. As an internal procedure in a super procedure of the local procedure.
3. As an internal procedure in a super procedure of the OpenEdge session.
4. As an external procedure file `add-record.p` or `add-record.r`.

Case 2: When Progress encounters a statement like the following:

```
RUN add-record('customer') IN my-proc-hdl.
```

Progress searches for add-record as follows:

1. As an internal procedure in my-proc-hdl.
2. As an internal procedure in a super procedure of my-proc-hdl.
3. As an internal procedure in a super procedure of the OpenEdge session.

Case 3: When Progress encounters a statement like the following:

```
add-record('customer').
```

Progress searches for add-record as follows:

1. As a user-defined function in the local procedure.
2. As a user-defined function in a super procedure of the local procedure.
3. As a user-defined function in a super procedure of the OpenEdge session.

Note: The rules of Case 3 apply whether or not the user-defined function's declaration (function prototype) includes the IN *proc-hdl* option. In Case 3, *proc-hdl* represents the local procedure. For more information on function prototypes of user-defined functions, see *OpenEdge Development: Progress 4GL Handbook*.

Search rules for SEARCH-TARGET

Case 4: A procedure, `main.p`, has added three super procedures, S1, S2, and S3 (in that order). Each of these super procedures has added its own super procedures, S1A, S1B, S2A, S2B, S3A, S3B. The procedure, add-record, exists in three places: in S1, in S2 where it contains a RUN SUPER statement, and in S2A.

When Progress encounters a statement like "`RUN add-record('customer').`", it searches for the add-record procedure:

1. As an internal procedure in the local procedure, `main.p`.
2. Then as an internal procedure in S3, and then in S2 where it is found.

If add-record was added with no *proc-search* value or with a *proc-search* value of SEARCH-SELF, when RUN SUPER is executed within add-record in S2, Progress starts searching in S2A, which is next in the search stack of the super procedure S2.

If add-record was added with a *proc-search* value of SEARCH-TARGET, when RUN SUPER is executed within add-record in S2, Progress will start searching in S1 which is next in the search stack of the local procedure, `main.p`.

Note: The search commences with the super procedure following *super-proc-hdl* in the local procedure's chain.

ADM-DATA attribute

An arbitrary string value associated with a persistent procedure.

Note: The ADM-DATA attribute is for use by the OpenEdge ADM only.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle, Buffer-field object handle, ProDataSet object handle, Data-relation object handle, Data-source object handle, Query object handle, SAX-attributes object handle, SAX-reader object handle, SOAP-header object handle, SOAP-header-entryref object handle, THIS-PROCEDURE system handle and all procedure handles

AFTER-BUFFER attribute

Returns the handle to the default buffer of the after-image table that corresponds to the buffer of the before-image table currently associated with this buffer handle.

Data type: HANDLE

Access: Readable

Applies to: Buffer object handle

See also BEFORE-BUFFER attribute

AFTER-ROWID attribute

Returns the ROWID of the row in the after-image table that is the current version of the row in the before-image table currently associated with this buffer handle. This row can be a new or modified row.

Data type: ROWID

Access: Readable

Applies to: Buffer object handle

This attribute is set to the Unknown value (?) for rows that have been deleted.

See also BEFORE-ROWID attribute

AFTER-TABLE attribute

Returns the handle of the after-image table that corresponds to the before-image table currently associated with this temp-table handle.

Data type: HANDLE

Access: Readable

Applies to: Temp-table object handle

See also BEFORE-TABLE attribute

ALLOW-COLUMN-SEARCHING attribute (Windows only)

Setting this attribute to TRUE allows column searching for browses.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

The default is FALSE for read-only static browses. The default is TRUE for dynamic browses and static updateable browses.

If ALLOW-COLUMN-SEARCHING is set to TRUE, the START-SEARCH and END-SEARCH events will be triggered when a search is initiated and completed.

ALWAYS-ON-TOP attribute (Windows only)

Indicates whether the window should remain on top of all windows, even windows belonging to other applications.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

This attribute differs from the TOP-ONLY attribute, which indicates that the window should remain on top of all windows of the OpenEdge session. Windows that have the ALWAYS-ON-TOP attribute set are always above TOP-ONLY windows in the z-order; they are also above dialog boxes in some cases.

A window cannot have both the TOP-ONLY and ALWAYS-ON-TOP attributes set. Setting the ALWAYS-ON-TOP attribute to TRUE will set the TOP-ONLY attribute to FALSE. The default value of the ALWAYS-ON-TOP attribute is FALSE.

AMBIGUOUS attribute

Indicates whether more than one record matched the FIND predicate.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle

If AMBIGUOUS is TRUE, the most recent unique find on the buffer failed because more than one record matched the FIND predicate. Otherwise, AMBIGUOUS is FALSE.

APPEND-CHILD( ) method

Appends a node as the last child node of this XML document or element node. Connects a node into the document structure after the node has been created with the CREATE-NODE( ) or CREATE-NODE-NAMESPACE( ) method, cloned with the CLONE-NODE( ) method, or disconnected with the REMOVE-NODE( ) method. This has no effect on the node reference.

If the node is already in the tree, it is disconnected from its present location and then connected at the specified location.

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
APPEND-CHILD( x-node-handle )
```

x-node-handle

The handle that represents the node to append to this XML document or element node. *x-node-handle* must refer to a node in this document. You cannot use APPEND-CHILD( ) to move a node from one document to another.

The following code fragment demonstrates creating a node in a tree with the name and value of a field:

```
. . .
hDoc:CREATE-NODE(hNoderef,bufField:NAME,"ELEMENT").
hNoderefParent:APPEND-CHILD(hNoderef).
hDoc:CREATE-NODE(hText,"","TEXT").
hText:NODE-VALUE = STRING(bufField:BUFFER-VALUE).
hNoderef:APPEND-CHILD(hText).
. . .
```

APPL-ALERT-BOXES attribute

Directs application messages to alert boxes or the default message area.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If the APPL-ALERT-BOXES attribute is TRUE, an application message produced by the MESSAGE statement is displayed in alert boxes rather than in the message area. The default value is FALSE.

APPL-CONTEXT-ID attribute

Returns the universally unique identifier (UUID) for the application context in effect for the current session, as a Base64 character string. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Data type: CHARACTER

Access: Readable

Applies to: AUDIT-CONTROL system handle

The value of this attribute is set by the SET-APPL-CONTEXT( ) method, and cleared by the CLEAR-APPL-CONTEXT( ) method. This value is recorded in all audit event records generated for this application context until you either clear the current application context or set a different application context.

If no application context is in effect, this method returns the Unknown value (?).

See also CLEAR-APPL-CONTEXT( ) method, SET-APPL-CONTEXT( ) method

APPLY-CALLBACK( ) method

Applies a callback procedure, which lets you execute a defined event without duplicating the event procedure definition.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Query object handle

Syntax

```
APPLY-CALLBACK( event-name )
```

event-name

The name of a defined event.

Use the SET-CALLBACK-PROCEDURE( ) method to associate an internal procedure with a callback for an object.

For more information on events, see the "Events Reference" section on page 2171.

See also GET-CALLBACK-PROC-CONTEXT( ) method, GET-CALLBACK-PROC-NAME( ) method, SET-CALLBACK-PROCEDURE( ) method

APPSERVER-INFO attribute

Connection parameter for the AppServer CONNECT( ) method.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

Valid only if LOCATOR-TYPE is "AppServer".

APPSERVER-PASSWORD attribute

Password parameter for the AppServer CONNECT( ) method.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

Valid only if LOCATOR-TYPE is "AppServer".

APPSERVER-USERID attribute

Userid parameter for the AppServer CONNECT( ) method.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

Valid only if LOCATOR-TYPE is "AppServer".

ASYNCHRONOUS attribute

Makes a dynamic invoke asynchronous. That is, the INVOKE( ) method with ASYNCHRONOUS set to TRUE does dynamically what the RUN statement with the ASYNCHRONOUS option does statically.

Note: If ASYNCHRONOUS is TRUE, the logic being invoked must reside on an AppServer.

Data type: LOGICAL

Access: Readable/Writable

Applies to: CALL object handle

Syntax

```
ASYNCHRONOUS [ = logical-expression ]
```

logical-expression

A LOGICAL expression which, if TRUE, makes the dynamic invoke asynchronous. The default is FALSE.

If ASYNCHRONOUS is TRUE, when the dynamic invoke returns successfully, it sets the ASYNC-REQUEST-HANDLE attribute to indicate an asynchronous-request object that provides information on this particular asynchronous request.

ASYNC-REQUEST-COUNT attribute

The number of active asynchronous requests for the specified procedure or AppServer.

Data type: INTEGER

Access: Readable

Applies to: Procedure object handle, Server object handle, THIS-PROCEDURE system handle

For a procedure handle, this attribute is only meaningful if PROXY and PERSISTENT are set to TRUE. In all other cases, this attribute returns zero (0).

Progress sets this attribute to one (1) on the following handles:

- A proxy persistent procedure handle created for an initial asynchronous request.
- The server handle of the AppServer where the asynchronous persistent procedure is instantiated.

Progress increments this attribute:

- On any proxy persistent procedure handle where an internal procedure defined in the specified remote persistent procedure context is executed asynchronously.
- On any server handle where an internal or external procedure is executed asynchronously on the specified AppServer.

Progress decrements this attribute as part of processing the PROCEDURE-COMPLETE event for one of the associated asynchronous requests. The attribute is decremented before any associated event procedure is executed in the context of a PROCESS EVENTS, WAIT-FOR, or other I/O-blocking statement.

ASYNC-REQUEST-HANDLE attribute

A handle to an asynchronous-request object providing information on an asynchronous invoke.

Note: Applies only if ASYNCHRONOUS is TRUE.

Data type: HANDLE

Access: Read Only

Applies to: CALL object handle

The default is the Unknown value (?).

ATTACH-DATA-SOURCE( ) method

Attaches a Data-source object to a temp-table buffer in a ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
buffer-handle:ATTACH-DATA-SOURCE ( datasource-hdl
    [ [ [ , pairs-list ], except-fields ], include-fields ] )
```

buffer-handle

A handle to the temp-table buffer in the ProDataSet object.

datasource-hdl

A handle to the Data-source object.

pairs-list

An optional character expression that evaluates to a comma-separated list of field pairs that specify different names in the Data-source object buffer and the ProDataSet object buffer using the following syntax:

```
"source-field1,dset-field1 [,source-fieldn,dset-fieldn ] ..."
```

You can use the ATTACHED-PAIRLIST attribute to retrieve this list of field name pairs.

except-fields

An optional character expression that evaluates to a comma-separated list of fields in the ProDataSet object buffer that will not be populated with data from the data source (that is, fields to exclude). Use this option when it is easier to specify fields to exclude rather than include. You can specify *except-fields* or *include-fields*, but not both.

include-fields

An optional character expression that evaluates to a comma-separated list of fields to include in the ProDataSet object buffer, as an alternative to specifying fields to exclude in *except-fields*. Use this option when it is easier to specify fields to include rather than exclude. You can specify *include-fields* or *except-fields*, but not both. If you specify *include-fields*, you must set *except-fields* to the Unknown value (?).

See also DETACH-DATA-SOURCE( ) method

ATTACHED-PAIRLIST attribute

Returns a comma-separated list of field name pairs for fields in a ProDataSet temp-table buffer that are mapped to corresponding fields in an attached Data-source object. This list includes only the field name pairs you specified with the most recently attached Data-source object.

Data type: CHARACTER

Access: Readable

Applies to: Buffer object handle

This list is formatted as a comma-separated list of field name pairs using the following syntax:

```
"source-field1,dset-field1 [,source-fieldn,dset-fieldn ] ..."
```

If the buffer is not part of a ProDataSet object, or the buffer does not have an attached Data-source object, or you did not specify a field name pair list when you attached the Data-source object, this attribute returns the Unknown value (?).

Use the DATA-SOURCE-COMPLETE-MAP attribute to retrieve a list of field name pairs for **all** fields in a ProDataSet temp-table buffer that are mapped to corresponding fields in an attached Data-source object.

See also ATTACH-DATA-SOURCE( ) method

ATTRIBUTE-NAMES attribute

Returns a comma-separated list of an element's attribute names. The attribute names are contained in the XML document. If the element does not have any attributes, the empty string ("") is returned.

Data type: CHARACTER

Access: Readable

Applies to: X-noderef object handle

If hNoderef is an element node with various attributes, and anames and bname are character program variables, the following example demonstrates listing all the attributes of the XML node:

```
anames = hNoderef:ATTRIBUTE-NAMES.
REPEAT j = 1 TO NUM-ENTRIES(anames):
  bname = ENTRY(j,anames).
  MESSAGE "attribute-name is" bname "value is"
       hNoderef:GET-ATTRIBUTE(bname).
END.
```

ATTR-SPACE attribute

This attribute has no effect. It is supported only for backward compatibility.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FILL-IN widget, TEXT widget

AUDIT-EVENT-CONTEXT attribute

The audit event context for a Client-principal object. Progress stores this application-defined audit context in the _Event-context field in the audit record created for an audit event generated by the SEAL( ) method, LOGOUT( ) method, and AUTHENTICATION-FAILED( ) method. If not specified, the _Event-context field in the audit record is left blank.

The value of this attribute cannot exceed 200 characters. You can also use this value as an alternate index for querying the audit event record.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the Client-principal object is sealed, this attribute is read-only.

See also AUTHENTICATION-FAILED( ) method, LOGOUT( ) method, SEAL( ) method

AUTHENTICATION-FAILED( ) method

Indicates that an unsealed Client-principal object could not be authenticated. This authentication failure signifies that the application or database user identity from the registered authentication domain is not authentic. Once invalidated, the Client-principal object's properties cannot be changed and the object cannot be sealed.

Note: An application can use this method to invalidate an unsealed Client-principal object for any reason.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
AUTHENTICATION-FAILED( [ reason ] )
```

reason

An optional character expression that specifies the reason for the authentication failure. For example, "Invalid user name or password". Progress sets the STATE-DETAIL attribute to this value.

If you call this method for a sealed Client-principal object, Progress generates a run-time error.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

Calling this method generates an audit event and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

Progress also sets the LOGIN-STATE attribute for the Client-principal object to "FAILED".

Use the LOGOUT( ) method to invalidate, or terminate access to, a **sealed** Client-principal object.

Example

The following code fragment illustrates how to use the AUTHENTICATION-FAILED( ) method:

```
DEF VAR hCP as HANDLE.
DEF VAR val-ok as LOGICAL.
.
.
.
CREATE CLIENT-PRINCIPAL hCp.
.
.
.
val-ok = hCP:AUTHENTICATION-FAILED("Invalid username or password").
```

See also

LOGIN-STATE attribute, LOGOUT( ) method, STATE-DETAIL attribute

AUTO-COMPLETION attribute (Windows only; Graphical interfaces only)

Specifies that the combo-box widget automatically complete keyboard input based on a potential match to items in the drop-down list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: COMBO-BOX widget

When the AUTO-COMPLETION attribute is TRUE, the widget's edit control compares the input to the items in the drop-down list. After each incremental character keystroke, the edit control searches through the items in the drop-down list for a potential match. If a potential match is found, the full item is displayed in the edit control. The automatically completed portion of the item is highlighted. You can replace the highlighted portion of the item by typing over it, or delete the highlighted portion of the item using the **DELETE** key or the **BACKSPACE** key. The default value is FALSE.

AUTO-DELETE attribute

Specifies whether a dynamic buffer and temp-table object associated with a ProDataSet object is automatically deleted when the ProDataSet object is deleted. Dynamic buffer and temp-table objects associated with a ProDataSet object are deleted when the ProDataSet object is deleted, by default.

Set this attribute to FALSE to prevent a dynamic buffer and temp-table from being automatically deleted when the associated ProDataSet object is deleted. The default value is TRUE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle

AUTO-DELETE-XML attribute

Determines whether the X-document object handle is deleted on a new web request. The default is YES.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

AUTO-END-KEY attribute

Directs Progress to apply the ENDKEY event to the current frame when a user chooses the button.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget

If the AUTO-END-KEY attribute is TRUE, Progress applies the ENDKEY event to the frame when the button is chosen. The default value is FALSE.

AUTO-ENDKEY is a synonym for the AUTO-END-KEY attribute.

AUTO-GO attribute

Directs Progress to apply the GO event to the current frame when a user chooses the button.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget

If the AUTO-GO attribute is TRUE, Progress applies the GO event to the frame when the button is chosen. The default value is FALSE.

AUTO-INDENT attribute

Specifies the text indentation behavior in the editor widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

If AUTO-INDENT is TRUE, each new line of text automatically indents to line up with the preceding line.

AUTO-RESIZE attribute (Graphical interfaces only)

Tells Progress how to resize a widget when the LABEL, FONT, or FORMAT attribute of the widget changes.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column), BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

If the AUTO-RESIZE attribute is TRUE, the widget automatically resizes when the LABEL, FONT or FORMAT attributes of the widget change. If AUTO-RESIZE is FALSE, the widget retains its original size.

The default value for this attribute is TRUE for widgets that are not explicitly sized when they are defined, and FALSE for explicitly sized widgets.

When the AUTO-RESIZE attribute is set to TRUE, Progress resizes button and toggle-box widgets with run-time changes to the LABEL attribute, and combo-box and fill-in field widgets with run-time changes to the FORMAT attribute.

This attribute resizes the following widgets with run-time changes to the FONT attribute:

- Browse-columns
- Buttons
- Combo boxes
- Editors
- Fill-ins
- Radio sets
- Selection lists
- Sliders
- Texts
- Toggle boxes

For browse-columns, if you set the browse-column's AUTO-RESIZE attribute to TRUE, Progress resizes the browse-column when a change occurs in the browse-column's font or in the font or text of the browse-column's label.

If the font of a browse-column grows such that the height needs to be increased, Progress increases the height of all cells in the browse-column, which increases the row height of the browse (because all rows have the same height). This might affect the DOWN, ROW-HEIGHT-CHARS, and ROW-HEIGHT-PIXELS attributes of the browse-column.

If the font of a browse-column decreases, Progress does not decrease the height of the rows, because the decrease might clip text in other columns.

Note: If the developer changes the size of the widget at run time by using the HEIGHT-CHARS, HEIGHT-PIXELS, WIDTH-CHARS, or WIDTH-PIXELS attribute, Progress resets AUTO-RESIZE to FALSE.

AUTO-RETURN attribute

Specifies the behavior that occurs when a user types the last allowable character in the widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column), FILL-IN widget

The FORMAT attribute controls the number of characters that a user can enter in the widget. By default, if the user attempts to enter more characters than the number allowed in the widget, Progress beeps and ignores characters. You can use the AUTO-RETURN attribute to alter this behavior only if the DATA-ENTRY-RETURN attribute of the SESSION handle is TRUE.

If DATA-ENTRY-RETURN and AUTO-RETURN are TRUE and a user types the last character in a field, a LEAVE event occurs and input focus moves to the next widget in the tab order. If the widget is the last widget in the tab order, a GO event occurs for the current frame. This behavior is the same as pressing **RETURN** in the field when the DATA-ENTRY-RETURN attribute is TRUE.

For browse-columns, if AUTO-RETURN is TRUE, when the user enters the last allowable character in a browse-cell, Progress behaves as if the user pressed the **RETURN** key.

AUTO-SYNCHRONIZE attribute

Indicates whether Progress automatically synchronizes a hierarchy of queries on a ProDataSet temp-table buffer.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle

Set to TRUE to synchronize the related buffers automatically. The default is FALSE.

When AUTO-SYNCHRONIZE is TRUE and a new row is placed in the buffer, the synchronize action occurs. The event handler is called when the buffer has a CREATE, DELETE, RELEASE, FIND, FOR-EACH, QUERY GET, or BUFFER-COPY run on it.

If the BUFFER-COPY is part of a FILL operation, a before-image operation (such as SAVE-ROW-CHANGES), or a deep-copy during parameter passing or COPY-TEMP-TABLE, then the synchronize action does not occur.

If you perform a manual FILL operation using BUFFER-COPY, you can prevent the query hierarchy from being synchronized unnecessarily by setting the AUTO-SYNCHRONIZE attribute to FALSE.

AUTO-VALIDATE attribute

Specifies when Progress will run the validation for a browse column.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (column)

If TRUE, Progress will run the validation for a browse column in the specified browser on LEAVE of the browse cell. If FALSE, Progress will run the validation only when code for a browse or browse-column specifically invokes the VALIDATE( ) method.

AUTO-ZAP attribute

Specifies what happens to the existing contents of the widget when the user types new information into the widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (cell), COMBO-BOX widget, FILL-IN widget

If AUTO-ZAP is TRUE, when the user begins typing in the field, the entire initial value is erased before the user's text appears. If AUTO-ZAP is FALSE, text entered by the user is inserted into existing text at the current cursor position in the field.

You can set AUTO-ZAP only when the fill-in or cell has input focus (its widget handle is equal to the FOCUS handle). Otherwise, AUTO-ZAP is TRUE when the user tabs or back-tabs into the field, highlighting text in the field. (When the user selects all text in the field, the same effect occurs without setting the AUTO-ZAP attribute.) AUTO-ZAP is FALSE when the user enters the field with the mouse pointer, positioning the text cursor in the field.

AVAILABLE attribute

Indicates whether a buffer contains a record.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle, Buffer-field object handle

For the buffer object handle, the AVAILABLE attribute corresponds to the AVAILABLE function. If the buffer contains a record, AVAILABLE is TRUE. Otherwise, AVAILABLE is FALSE.

Generally, a buffer-field object handle corresponds to a field returned in a query buffer. However, this field can be excluded from the query using a field list. In this case, if you try to read the BUFFER-VALUE attribute on the associated buffer-field object handle, Progress returns an error indicating that the corresponding field is missing from the query buffer. You can use the AVAILABLE attribute to test whether the corresponding field was included or excluded from the query.

Depending on its return value, the AVAILABLE attribute indicates one of the following conditions when applied to the buffer-field object:

- **TRUE** — The query buffer has a record with a field available that corresponds to this buffer-field object handle.
- **FALSE** — The query buffer has a record with the field missing that corresponds to this buffer-field object handle.
- **Unknown value (?)** — The query buffer associated with this buffer-field object handle has no record.

AVAILABLE-FORMATS attribute

A comma-separated list of names that specify the formats available for the data currently stored in the clipboard.

Data type: CHARACTER

Access: Readable

Applies to: CLIPBOARD system handle

If there are no formats available, the attribute returns the Unknown value (?). The supported formats include:

- **PRO_TEXT** — Specifies the standard text format on your system (CF_TEXT in Windows).
- **PRO_MULTIPLE** — Specifies that the data in the clipboard contains tab or newline characters, and thus can be read as multiple items.

For more information, see the reference entry for the CLIPBOARD system handle.

BACKGROUND attribute

Specifies widget handle for the background iteration of the frame or dialog box.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

BASE-ADE attribute

Sets the location of the ADE r-code directory. When set, Progress adds the directory, followed by all the procedure libraries in the directory, to the PROPATH.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

You can also specify the ADE r-code location using the ADE R-code Location (-baseADE) startup parameter. For more information about the ADE R-code Location (-baseADE) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

With the BASE-ADE attribute and the ADE R-code Location (-baseADE) startup parameter, you can have multiple versions of ADE r-code on the same machine and easily switch between them.

Notes

- If -baseADE is not specified at startup, SESSION:BASE-ADE has the Unknown value (?) until it is set.
- When SESSION:BASE-ADE is set, Progress adds the directory, followed by all of the procedure libraries in the directory to PROPATH. If the directory or any of the procedure libraries are already on PROPATH, Progress does not add them.
- When SESSION:BASE-ADE is set, Progress removes all PROPATH entries representing the current ADE r-code directory and procedure libraries before adding the new PROPATH entries. Progress adds the new PROPATH entries at the location where it removed the previous entries. Progress only removes PROPATH entries that it added. For example, if $DLC/gui/adecomm.pl is part of the PROPATH, it remains on the PROPATH after BASE-ADE is set to a directory other than $DLC/gui.
- If BASE-ADE is set to an empty string, Progress removes whatever it added to PROPATH.
- Progress does not remove the ADE r-code directory or any of the procedure libraries in that directory from PROPATH, even if the PROPATH statement does not contain them. These entries are part of the base PROPATH. If -baseADE or SESSION:BASE-ADE is used, the directory and procedure libraries that Progress adds are part of the base PROPATH and remain part of the PROPATH even if the PROPATH statement does not contain them.
- SESSION:BASE-ADE modifies PROPATH. If the old PROPATH contains a procedure library that is not in the new PROPATH, Progress automatically closes the procedure library as long as there are no procedures from the library running.

BASIC-LOGGING attribute

Turns on `QryInfo` logging for an individual query.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Query object handle

Progress does not provide `QryInfo` logging for individual queries at logging level 2 (Basic), by default. You must use the BASIC-LOGGING attribute to turn on logging for an individual query when the logging level of the `QryInfo` log entry type is set to level 2 (Basic). If the logging level of the `QryInfo` log entry type is level 3 (Verbose) or higher, then Progress ignores any BASIC-LOGGING attribute setting and logs query information for all queries.

To set the logging level of the `QryInfo` log entry type, use the LOG-ENTRY-TYPES attribute or the Log Entry Types (`-logentrytypes`) startup parameter with the logging level option.

To turn on logging for an individual query when the logging level is set to 2 (Basic), you must set this attribute to `TRUE` before a query starts. For a dynamically opened query, this is before the QUERY-PREPARE( ) method. For a statically opened query, this is before the OPEN QUERY statement. If you set this attribute to `TRUE` after a query starts, Progress does not provide logging for that query.

To turn off logging for an individual query when the logging level is set to 2 (Basic), you must set this attribute to `FALSE`. If you set this attribute to `FALSE` before a query completes, Progress does not write query statistics to the log.

Whenever you turn on or turn off logging for an individual query using this attribute, Progress writes a log entry to the log file indicating the query ID, the query object handle, and the name of the query.

For more information about the Log Entry Types (`-logentrytypes`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

For more information about query logging, see *OpenEdge Development: Debugging and Troubleshooting*.

BATCH-MODE attribute

Indicates whether the current OpenEdge session is running in batch mode or interactive mode.

Data type: LOGICAL

Access: Readable

Applies to: SESSION system handle

If BATCH-MODE is TRUE, the current session is running with Batch (`-b`) parameter in batch or background mode.

See also BATCH-SIZE attribute, FILL( ) method, LAST-BATCH attribute

BATCH-SIZE attribute

The maximum number of ProDataSet temp-table rows to retrieve in each FILL operation. The default value is zero (which retrieves all rows that satisfy the associated query).

Note: If you specify a batch size for a ProDataSet temp-table that is a child of a relation, Progress restarts the BATCH-SIZE counter for each parent record (as opposed to once per temp-table).

Data type: INTEGER

Access: Readable/Writeable

Applies to: Temp-table object handle

This attribute is not marshalled between the client and the AppServer (unlike the LAST-BATCH attribute).

See also BATCH-MODE attribute, FILL( ) method, LAST-BATCH attribute

BEFORE-BUFFER attribute

Returns the handle to the default buffer of the before-image table that corresponds to the buffer of the after-image table currently associated with this buffer handle.

Data type: HANDLE

Access: Readable

Applies to: Buffer object handle

See also AFTER-BUFFER attribute

BEFORE-ROWID attribute

Returns the ROWID of the row in the before-image table that corresponds to the row in the after-image table currently associated with this buffer handle.

Data type: ROWID

Access: Readable

Applies to: Buffer object handle

This attribute is set to the Unknown value (?) for row that have not changed.

See also AFTER-ROWID attribute

BEFORE-TABLE attribute

Returns the handle of the before-image table that corresponds to the after-image table currently associated with this temp-table handle.

Data type: HANDLE

Access: Readable

Applies to: Temp-table object handle

See also AFTER-TABLE attribute

BEGIN-EVENT-GROUP( ) method

Indicates (and records) the beginning of a group of related audit events in the current session. Audit event groups are used to group a series of related application and database audit events in one or more connected audit-enabled databases whose current audit policy has this audit event enabled.

This method returns a Base64 character string that specifies the universally unique identifier (UUID) of the primary index for all audit event records generated by this method for this audit event group. This UUID is recorded in all subsequent audit event records until you either end this audit event group or begin a different audit event group. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Return type: CHARACTER

Applies to: AUDIT-CONTROL system handle

Syntax

```
BEGIN-EVENT-GROUP( event-context [, event-detail [, user-detail ] ] )
```

event-context

A character expression that specifies the context for the audit event. The value of this expression cannot exceed 200 characters. You can also use this value as an alternate index for querying the audit event record.

If you specify the Unknown value (?), Progress generates a run-time error.

event-detail

An optional character expression that specifies additional audit detail. The value of this expression cannot exceed 10,000 characters.

user-detail

An optional character expression that specifies additional user detail. The value of this expression cannot exceed 10,000 characters.

The UUID is saved as the EVENT-GROUP-ID attribute value for each connected audit-enabled database.

There can be only one active event group per session at any one point in time. To set a different event group for the session, you can:

- Call the END-EVENT-GROUP( ) method, to end the current event group, and then call the BEGIN-EVENT-GROUP( ) method to begin the new event group.
- Call the BEGIN-EVENT-GROUP( ) method to begin the new event group. If there is an existing event group in effect, Progress ends the existing event group before beginning the new event group.

Calling this method generates an audit event, and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

Example The following code fragment illustrates how to use the BEGIN-EVENT-GROUP( ) method:

```
DEF VAR name as CHAR.
DEF VAR ctx-id as CHAR.
DEF VAR grp-id as CHAR.
.
.
.
grp-id = AUDIT-CONTROL:BEGIN-EVENT-GROUP("Payroll app", "tax calculations",
name).
ctx-id = AUDIT-CONTROL:SET-APPL-CONTEXT("Payroll app", "federal tax
calculation", name).
.
.
.
AUDIT-CONTROL:LOG-AUDIT-EVENT(34122, "payroll.fed.tax.nh").
.
.
.
ctx-id = AUDIT-CONTROL:SET-APPL-CONTEXT("Payroll app", "fica calculation",
name).
.
.
.
AUDIT-CONTROL:LOG-AUDIT-EVENT(34123, "payroll.fed.tax.ma").
.
.
.
AUDIT-CONTROL:CLEAR-APPL-CONTEXT.
AUDIT-CONTROL:END-EVENT-GROUP.
```

See also END-EVENT-GROUP( ) method, EVENT-GROUP-ID attribute

BGCOLOR attribute (Graphical interfaces only)

Specifies the color number for the background color of the widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

For a rectangle, if the FILLED attribute is TRUE, BGCOLOR specifies the color of the region inside the border of the rectangle.

For a browse cell, BGCOLOR specifies the color of a specific cell in the view port. You can set this color only as the cell appears in the view port during a ROW-DISPLAY event.

BLANK attribute

Suppresses the display of sensitive data in a field.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FILL-IN widget, TEXT widget

If the BLANK attribute is TRUE, any current value or characters typed in the fill-in are not echoed to the screen. The default value for this attribute is FALSE.

BLOCK-ITERATION-DISPLAY attribute

Specifies if the Frame phrase of the frame contains the NO-HIDE option or if the frame has multiple iterations (is a DOWN frame).

Data type: LOGICAL

Access: Readable

Applies to: FRAME widget

The BLOCK-ITERATION-DISPLAY attribute returns TRUE if the NO-HIDE option is specified in a frame phrase for the frame or the frame has multiple iterations.

BORDER-BOTTOM-CHARS attribute

The thickness, in character units, of the border at the bottom of the frame or dialog box.

Data type: DECIMAL

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-BOTTOM-PIXELS attribute

The thickness, in pixels, of the border at the bottom of the frame or dialog box.

Data type: INTEGER

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-LEFT-CHARS attribute

The thickness, in character units, of the border at the left side of the frame or dialog box.

Data type: DECIMAL

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-LEFT-PIXELS attribute

The thickness, in pixels, of the border at the left side of the frame or dialog box.

Data type: INTEGER

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-RIGHT-CHARS attribute

The thickness, in character units, of the border at the right side of the frame or dialog box.

Data type: DECIMAL

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-RIGHT-PIXELS attribute

The thickness, in pixels, of the border at the right side of the frame or dialog box.

Data type: INTEGER

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-TOP-CHARS attribute

The thickness, in character units, of the border at the top of the frame or dialog box.

Data type: DECIMAL

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BORDER-TOP-PIXELS attribute

The thickness, in pixels, of the border at the top of the frame or dialog box.

Data type: INTEGER

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute returns zero if the BOX attribute for the widget is FALSE.

BOX attribute (Windows only; Graphical interfaces only)

Indicates whether the widget has a graphical border around it.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget, FRAME widget

If the BOX attribute is FALSE, the widget does not have a border. You can set this attribute only before the widget is realized.

For editors, BOX has no effect on the size of the editor.

BOX-SELECTABLE attribute (Graphical interfaces only)

Indicates whether box-selection direct manipulation events for the frame or dialog box are enabled or disabled.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

If the BOX-SELECTABLE attribute is TRUE, then the START-BOX-SELECTION and END-BOX-SELECTION direct manipulation events are enabled for the frame or dialog box. This allows the user to select one or more widgets in the frame or dialog box by stretching a select box around the widgets. The SELECTABLE attribute must be TRUE for at least one widget in the frame or dialog box for this attribute to be effective. Otherwise, the user can stretch a select box, but without any effect on the widgets in the frame or dialog box. For more information on box selection, see the chapter on direct manipulation in *OpenEdge Development: Progress 4GL Handbook*.

BUFFER-CHARS attribute (Character interfaces only)

The number of characters a user can enter on each line of the editor.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget

You can set this attribute only before the editor widget is realized. The value must be an integer expression that is equal to or greater than the value specified by the WIDTH-CHARS or INNER-CHARS attributes. If greater, horizontal scrolling is enabled.

When the last character is typed on a line in the editor, the text input cursor automatically wraps to the next line. This attribute can also set the word wrap margin for the WORD-WRAP attribute. For more information, see the WORD-WRAP attribute reference entry.

BUFFER-COMPARE( ) method

This method does a rough compare of any common fields, determined by name, data type, and extent-matching, between the source buffer and the target buffer. The resulting logical value is either TRUE or FALSE as a whole. A single field that does not compare causes the entire buffer to return FALSE. If there are fields in one buffer that do not exist in the other, they are ignored.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-COMPARE( source-buffer-handle [ , mode-exp [ , except-list
     [ , pairs-list [ , no-lobs ] ] ] ] )
```

source-buffer-handle

An expression that evaluates to a buffer handle.

mode-exp

If *mode-exp* is given, it must evaluate to either “binary” or “case-sensitive” to provide that type of comparison.

except-list

A character expression that evaluates to a comma-separated list of field names to be excluded from the compare.

pairs-list

A character expression that evaluates to a comma-separated list of field-name pairs to be compared.

You can specify an array element as one or both of the fields. This lets you compare a field or array element in one buffer to a field or array element in the other buffer, when the two fields do not have the same name. The order within each field-name pair does not matter; each pair must contain one field name from the source and one field name from the target.

You can also compare one entire array to another by specifying its name without a subscript.

no-lobs

A logical expression indicating whether to ignore BLOB and CLOB fields in the compare. If TRUE, BLOB and CLOB fields are ignored during the compare. If FALSE, BLOB and CLOB fields are compared along with the other fields. The default value is FALSE (that is, BLOB and CLOB fields are included in the compare).

Note: You cannot use the BUFFER-COMPARE( ) method to compare records that contain CLOB fields, unless one or both of the corresponding fields contain the Unknown value (?); Progress generates a run-time error. However, you can convert CLOB fields to LONGCHAR values and use the EQ, GE, GT, LE, LT, and NE comparison operators, or the COMPARE function, to compare the LONGCHAR values.

If you want to compare BLOB fields only, you can set this option to FALSE and use the *except-list* option to exclude CLOB fields from the compare.

The following example fragment does a binary compare of two fields, one from each buffer:

```
BUFFER-COMPARE(bh2,"binary","cust-sales-rep,sales-rep").
```

Notes

- When comparing buffers in a ProDataSet object, Progress checks as to whether the BUFFER-COMPARE( ) method satisfies the following two requirements:
  - The compare is between a buffer on a Data-source object table and the corresponding ProDataSet temp-table buffer. This means the operation can use any buffer for the data source database table, but only the default buffer for the ProDataSet temp-table.
  - There are no *except-list* or *pairs-list* arguments for the BUFFER-COMPARE( ) method.

  If these two requirements are satisfied, the BUFFER-COMPARE( ) method identifies the fields to compare based on the *pairs-list* argument specified in the ATTACH-DATA-SOURCE( ) method for the Data-source object, if any, along with either the *except-list* or *include-list* arguments, if any. Because the ATTACH-DATA-SOURCE( ) method already allows you to define a field mapping between a Data-source object buffer and a ProDataSet temp-table buffer, as well as a list of fields to include or exclude from the operation, you do not need to specify these in the BUFFER-COMPARE( ) method.
- When comparing records that contain BLOB fields, Progress performs a binary comparison on the BLOB data associated with the source and target records.
- Use the *no-lobs* option with the BUFFER-COMPARE( ) method to ignore large object data when comparing records that contain BLOB or CLOB fields. You can also use the *except-list* option to exclude BLOB and CLOB fields from the compare.

BUFFER-COPY( ) method

This method copies any common fields, determined by name, data type, and extent-matching, from the source buffer to the receiving buffer. If there are fields in one buffer that do not exist in the other, they are ignored. This method is used to accommodate temp-tables of joins.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-COPY( source-buffer-handle
   [ , except-list [ , pairs-list [ , no-lobs ] ] ] )
```

source-buffer-handle

An expression that evaluates to the source buffer handle.

except-list

A character expression that evaluates to a comma-separated list of field names to be excluded from the copy.

pairs-list

A character expression that evaluates to a comma-separated list of field-name pairs to be copied.

You can also specify an array element as one or both of the fields. This lets you copy a field or array element from one buffer to a field or array element in the other buffer, when the two fields do not have the same name. The order within each field-name pair does not matter; each pair must contain one field name from the source and one field name from the target.

You can also copy one entire array to another by specifying its name without a subscript.

no-lobs

A logical expression indicating whether to ignore BLOB and CLOB fields in the copy. If TRUE, BLOB and CLOB fields are ignored during the copy. If FALSE, BLOB and CLOB fields are copied along with the other fields. The default value is FALSE (that is, BLOB and CLOB fields are included in the copy).

The following example fragment copies the customer table to the buffer, *bh*, except that `customer.sales-rep` is copied to a field called *`cust-sales-rep`* in the buffer:

```
bh:BUFFER-COPY(buffer customer:handle,?,"cust-sales-rep,sales-rep").
```

Note

- When copying buffers in a ProDataSet object, Progress checks as to whether the BUFFER-COPY( ) method satisfies the following two requirements:
  - The copy is between a buffer on a Data-source object table and the corresponding ProDataSet temp-table buffer. This means the operation can use any buffer for the data source database table, but only the default buffer for the ProDataSet temp-table.
  - There are no *`except-list`* or *`pairs-list`* arguments for the BUFFER-COPY( ) method.

  If these two requirements are satisfied, the BUFFER-COPY( ) method identifies the fields to copy based on the *`pairs-list`* argument specified in the ATTACH-DATA-SOURCE( ) method for the Data-source object, if any, along with either the *`except-list`* or *`include-list`* arguments, if any. Because the ATTACH-DATA-SOURCE( ) method already allows you to define a field mapping between a Data-source object buffer and a ProDataSet temp-table buffer, as well as a list of fields to include or exclude from the operation, you do not need to specify these in the BUFFER-COPY( ) method.
- When copying records that contain a BLOB or CLOB field, Progress copies the object data associated with the source record to the target record. If the BLOB or CLOB field in the source record contains the Unknown value (?), Progress stores the Unknown value (?) in the BLOB or CLOB field of the target record. If the target record already has object data associated with it, Progress deletes that object data before copying the new object data.
- Use the *`no-lobs`* option with the BUFFER-COPY( ) method to ignore large object data when copying records that contain BLOB or CLOB fields. More specifically:
  - When you copy a source record to a new target record, Progress sets the value of the BLOB or CLOB field in the target record to the Unknown value (?).
  - When you copy a source record to an existing target record, Progress does not change the value of the BLOB or CLOB field in the existing target record.

  You can also use the *`except-list`* option to exclude BLOB and CLOB fields from the copy.

BUFFER-CREATE( ) method

Creates a record, sets fields to their default values, and moves a copy of the record into the buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-CREATE ( )
```

Note: The BUFFER-CREATE method corresponds to the CREATE statement.

BUFFER-DELETE( ) method

Deletes a record from the record buffer and from the database.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-DELETE ( )
```

Notes

- The BUFFER-DELETE method corresponds to the DELETE statement.
- If the table has delete validation—that is, if the table specifies an expression that must be true before the record is deleted—the record is not deleted, because the validation expression, normally applied at compile time, cannot be applied fully at run time.

BUFFER-FIELD attribute

The handle of the browse column's buffer-field.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: BROWSE widget (column)

If the browse column does not have a corresponding buffer field, the Unknown value (?) will be returned.

BUFFER-FIELD( ) method

Returns a handle to a particular field in the buffer.

Return type: WIDGET-HANDLE

Applies to: Buffer object handle

Syntax

```
BUFFER-FIELD ( field-number | field-name )
```

field-number

An INTEGER expression representing the sequence number of the field in the buffer.

field-name

A CHARACTER string expression representing the name of the field in the buffer.

BUFFER-HANDLE attribute

The handle of the buffer object to which the buffer-field belongs.

Data type: HANDLE

Access: Readable

Applies to: Buffer-field object handle

BUFFER-LINES attribute (Character interfaces only)

The number of lines a user can enter into the editor.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget

You can set this attribute only before the editor widget is realized. The value must be an integer expression that is equal to or greater than the value specified by the HEIGHT-CHARS or INNER-LINES attributes. If equal, vertical scrolling is disabled.

By default, Progress does not limit the number of enterable lines (although system limits may apply).

BUFFER-NAME attribute

The name of the buffer object to which the buffer-field object belongs.

Data type: CHARACTER

Access: Readable

Applies to: Buffer-field object handle

BUFFER-RELEASE( ) method

Releases a record from a buffer object. The BUFFER-RELEASE method corresponds to the RELEASE statement.

Note: To delete the buffer object, use the DELETE OBJECT statement.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-RELEASE ( )
```

If a record has been modified, this method causes a WRITE event and executes all related WRITE triggers.

If successful, this method returns TRUE. Otherwise, it returns FALSE. If the validation fails on a newly-created record, this method returns FALSE and raises the ERROR condition.

BUFFER-VALIDATE( ) method

Verifies that a record in a buffer object complies with mandatory field and unique index definitions. The BUFFER-VALIDATE( ) method corresponds to the VALIDATE statement.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
BUFFER-VALIDATE ( )
```

If a field or table has been modified, this method causes a WRITE event and executes all related WRITE triggers.

If successful, this method returns TRUE. Otherwise, it returns FALSE. If the validation fails on a newly-created record, this method returns FALSE and raises the ERROR condition.

The record is not released and the lock status remains the same as before the BUFFER-VALIDATE( ).

BUFFER-VALUE attribute

The current value of a buffer-field object. If you modify the BUFFER-VALUE attribute, Progress sets the buffer-field object to the new value.

Data type: The data type of the corresponding buffer-field

Access: Readable/Writeable

Applies to: Buffer-field object handle

Syntax

```
BUFFER-VALUE [ ( i ) ]
```

i

An INTEGER expression representing a subscript, for fields that have extents.

The syntax for retrieving the value of a buffer-field object using a dynamic reference to a table field in a dynamic ProDataSet, Temp-table, Query, or Buffer object can be awkward, especially when you know the table and field names at compile time. Progress provides a simpler way to express the same syntax, only in a short-hand form. For example, following is the typical syntax for referring to the `Cust-num` buffer field in the `Customer` table through a ProDataSet handle:

```
hDSet:GET-BUFFER-HANDLE("ttcust"):BUFFER-FIELD("Cust-num"):BUFFER-VALUE
```

Following is the short-hand form of the same syntax:

```
hDSet::ttcust::Cust-num
```

See also LITERAL-QUESTION attribute

BYTES-READ attribute

Returns the number of bytes read from the socket via the last READ( ) method. If the last READ( ) method failed, this attribute will return 0.

Data type: INTEGER

Access: Readable

Applies to: Socket object handle

BYTES-WRITTEN attribute

Returns the number of bytes written to the socket via the last WRITE( ) method. If the last WRITE( ) method failed, this attribute will return 0.

Data type: INTEGER

Access: Readable

Applies to: Socket object handle

CACHE attribute

Specifies how many records a NO-LOCK query should hold in memory.

Data type: INTEGER

Access: Readable/Writeable

Applies to: Query object handle

Note: The CACHE attribute corresponds to the CACHE option of the DEFINE QUERY statement.

CALL-NAME attribute

The name of one of the following:

- An external procedure, internal procedure, or user-defined function you want to invoke dynamically.
- An attribute you want to get or set dynamically.
- A method you want to run dynamically.

Data type: CHARACTER

Access: Readable/Writable

Applies to: CALL object handle

Syntax

```
CALL-NAME [ = char-expression ]
```

char-expression

A CHARACTER expression indicating the name of the procedure, user-defined function, or attribute. The default is the Unknown value (?).

If *char-expression* is a procedure, the rules for finding the procedure are the same as those of the RUN statement, including the treatment of SUPER. However, the actual instance of the file, procedure name, or attribute is not determined until the dynamic invoke is executed.

If *char-expression* is a user-defined function, an attribute, or a method, the rules for finding the function or attribute are the same as those for finding a function or attribute invoked statically.

CALL-TYPE attribute

The type of call, which must be the dynamic version of one of the following:

- Invoking an external or internal procedure.
- Invoking a user-defined function.
- Getting an attribute.
- Setting an attribute.
- Invoking a method.

Data type: INTEGER

Access: Readable/Writable

Applies to: CALL object handle

You can use a keyword constant or an integer. Table 54 lists the keyword constants and integers you can use.

Table 54: Keyword constants for the CALL-TYPE attribute

Keyword constant	Integer
PROCEDURE-CALL-TYPE	1
FUNCTION-CALL-TYPE	2
GET-ATTR-CALL-TYPE	3
SET-ATTR-CALL-TYPE	4

The default is PROCEDURE-CALL-TYPE.

To get the value of an attribute or to invoke a method, set CALL-TYPE to GET-ATTR-CALL-TYPE.

To set the value of *object:attribute* to the first parameter set by the SET-PARAMETER( ) method, set CALL-TYPE to SET-ATTR-CALL-TYPE, as in the following fragment:

```
/* set SESSION:NUMERIC-FORMAT to "european" */

hCall:IN-HANDLE = "session".
HCALL:CALL-TYPE = SET-ATTR-CALL-TYPE.
hCall:CALL-NAME = "numeric-format".
hCall:NUM-PARAMETERS = 1.
hCall:SET-PARAMETER( 1, "CHAR", "INPUT", "european").
hCall:INVOKE.
```

Note: Progress Software Corporation recommends that you do not set the CALL object's CALL-TYPE attribute to SET-ATTR-CALL-TYPE to set a BUFFER-FIELD object's BUFFER-VALUE attribute, since there in no way to run triggers for the target field.

CANCEL-BREAK( ) method

Cancels a breakpoint from a debugging session.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
CANCEL-BREAK( [ procedure [ , line-number ] ] )
```

procedure

A character expression that specifies the name of the procedure for which you want to cancel a breakpoint. The specified procedure does not have to exist at the time the breakpoint is cancelled. If you do not specify *procedure*, the method cancels any breakpoint set on the line immediately following the current line. (This is different from the SET-BREAK( ) method, which sets a breakpoint on the next executable line.)

line-number

An integer expression that specifies the line number in *procedure* (based at line 1 of the debug listing) where you want to cancel the breakpoint. A positive integer greater than or equal to 1 represents a line number in the specified *procedure* file. Zero (0) or a negative integer value represents the first executable line of the main procedure block in the specified *procedure* file. If you do not specify *line-number*, the method cancels the breakpoint at the first executable line of *procedure* file.

If you invoke DEBUGGER:CANCEL-BREAK (*procedure* , *line-number*) on the same line that is specified by *procedure* and *line-number*, the existing breakpoint on the specified line occurs the first time it is executed. The breakpoint is cancelled only on the second and succeeding executions of the line.

Note: To use this method, you must have the Application Debugger installed in your OpenEdge environment.

For more information, see the reference entry for the DEBUGGER system handle.

CANCEL-BUTTON attribute

A button widget in the frame or dialog box to receive the CHOOSE event when a user cancels the current frame or dialog box by pressing the **ESC** key.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

Any other action normally associated with the **ESC** key is not performed. The **ESC** key is any key associated with the **ESC** key label.

The CANCEL-BUTTON attribute for frames is not supported in character mode.

Note: If the user presses ESCAPE in a frame that has no such cancel button and the frame is part of a frame family, Progress applies the CHOOSE event to the first cancel button it finds within the frame family in random order.

CANCEL-REQUESTS( ) method

For a state-reset, state-aware, or stateless AppServer, this method raises a STOP condition in the context of the currently running asynchronous request and purges the send queue of any asynchronous requests that have not been executed on the specified AppServer.

For a state-free AppServer, this method raises a STOP condition for all currently running asynchronous requests, and purges the send queue of any asynchronous requests that have not been executed on the specified AppServer.

For Web services, this method terminates the connection to all currently running asynchronous requests and purges the send queue of all asynchronous requests that have not been executed on the specified Web service.

Return type: LOGICAL

Applies to: Server object handle

Syntax

```
CANCEL-REQUESTS( )
```

After executing this method, at the next I/O-blocking state (or on executing the PROCESS EVENTS statement) event procedures execute for the following asynchronous requests:

- All requests that were complete when this method executed but whose event procedures have not been run.
- All currently running requests that were stopped in response to this method.
- All requests that were purged from the send queue, and never run.

This method returns FALSE when the server handle is not in the connected state. (See the CONNECTED( ) method). Otherwise, this method returns TRUE.

CANCELLED attribute

Indicates if the asynchronous request was cancelled using either the CANCEL-REQUESTS( ) method or the DISCONNECT( ) method on the associated server handle.

Data type: LOGICAL

Access: Readable

Applies to: Asynchronous request object handle

If set to TRUE, the request is cancelled. Otherwise, it is pending or complete (see the COMPLETE attribute).

CAN-CREATE attribute

Indicates whether the Progress user has permission to insert into the database the record associated with a buffer.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle

For information about checking permissions at compile time and run time, see *OpenEdge Deployment: Managing 4GL Applications*.

CAN-DELETE attribute

Indicates whether the Progress user has permission to delete from the database the record associated with a buffer.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle

For information about checking permissions at compile time and run time, see *OpenEdge Deployment: Managing 4GL Applications*.

CAN-READ attribute

Indicates whether the Progress user has permission to read the record associated with a buffer or buffer-field.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle, Buffer-field object handle

For information about checking permissions at compile time and run time, see *OpenEdge Deployment: Managing 4GL Applications*.

CAN-WRITE attribute

Indicates whether the Progress user has permission to modify the record associated with a buffer or buffer-field.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle, Buffer-field object handle

For information about checking permissions at compile time and run time, see *OpenEdge Deployment: Managing 4GL Applications*.

CAREFUL-PAINT attribute

Indicates whether overlapping widgets in a 3D frame will refresh (repaint) carefully but more slowly (TRUE), or quickly, but possibly not as carefully (FALSE).

The CarefulPaint setting in the Startup section of the `progress.ini` file is used to determine the initial setting of the CAREFUL-PAINT attribute. The default value is TRUE.

You can set this frame attribute at any time.

Data type: LOGICAL

Access: Readable/Writable

Applies to: FRAME widget

CASE-SENSITIVE attribute

Indicates whether a buffer-field is case-sensitive.

Data type: LOGICAL

Access: Readable

Applies to: Buffer-field object handle

CENTERED attribute

Indicates whether Progress automatically centers the frame in a window.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

The default value for this attribute is FALSE. When you set this attribute from FALSE to TRUE, the values of the COLUMN, ROW, X, and Y attributes for the frame change to reflect the new location of the frame. Setting the CENTERED attribute from TRUE to FALSE has no meaning and results in an error message.

CHARSET attribute

The current setting of the Character Set (`-charset`) parameter.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

The CHARSET attribute returns a value that specifies the character set used for Progress data — "`iso8859-1`" or "`undefined`". The value is set by the Character Set (`-charset`) parameter.

This attribute is obsolete. See the CPINTERNAL attribute.

CHECKED attribute

The display state for a toggle box or a toggle-box menu item .

Data type: LOGICAL

Access: Readable/Writeable

Applies to: MENU-ITEM widget, TOGGLE-BOX widget

When this attribute is TRUE, the center of the toggle is filled to represent the “on” state of the value associated with the widget. Setting this attribute to FALSE removes the fill from the center of the toggle to represent the “off” state for the value associated with the widget.

CHILD-BUFFER attribute

Returns the buffer handle of the child member of the data-relation object.

Data type: HANDLE

Access: Readable

Applies to: Data-relation object handle

CHILD-NUM attribute

Returns the relative number assigned to this XML node among its siblings. XML nodes that have the same parent are called siblings, and are numbered from 1 to the number of siblings.

Data type: INTEGER

Access: Readable

Applies to: X-noderef object handle

The Unknown value (?) is returned if the node reference does not refer to an element node, or if the node is an XML document node.

The following example demonstrates the use of the CHILD-NUM attribute:

```
my-index = hNoderef:CHILD-NUM.
```

CLASS-TYPE attribute

Returns the class type of the most recently compiled class definition (`.cls`) file. If the most recently compiled file was not a class definition file, this attribute returns the empty string ("").

Data type: CHARACTER

Access: Readable

Applies to: COMPILER system handle

CLEAR( ) method

This method performs the following actions:

- Resets each attribute of a CALL object to its default value, which lets you reuse a CALL object.
- Removes all elements from a dynamic ProDataSet object including buffers and relations. That is, it restores the state of the dynamic ProDataSet object to what it was immediately after the CREATE DATASET statement.
- Initializes the internal state of the Application Debugger.
- Clears temp-table definitions and removes temp-table data.

Return type: LOGICAL

Applies to: CALL object handle, ProDataSet object handle, DEBUGGER system handle, Temp-table object handle

Syntax

```
CLEAR ( )
```

For the Application Debugger system handle:

- This method has no visible effect on the Debugger window.
- If the Debugger is initialized, this method returns TRUE. Otherwise, it returns FALSE with no effect.

Note: To use this method with the DEBUGGER system handle, you must have the Application Debugger installed in your OpenEdge environment.

For more information on using this method for the Debugger, see the reference entry for the DEBUGGER system handle and *OpenEdge Development: Debugging and Troubleshooting*.

For the CALL object handle, this method resets each attribute of a CALL object to its default value, which lets you reuse a CALL object.

Note: To reset just the parameters, set the NUM-PARAMETERS attribute to zero.

For the temp-table object handle:

- This method empties the temp-table and removes all its definitional data (field and index definitions and pending saved data). This puts the temp-table object into the CLEAR state, as opposed to the UNPREPARED or PREPARED state.
- Calling any method after this one changes the state to UNPREPARED.

CLEAR-APPL-CONTEXT( ) method

Clears the application context for the current session.

Return type: LOGICAL

Applies to: AUDIT-CONTROL system handle

Syntax

```
CLEAR-APPL-CONTEXT( )
```

After calling this method, the APPL-CONTEXT-ID attribute is cleared for all connected audit-enabled databases and is no longer recorded in audit event records for this application context.

There can be only one active application context per session at any one point in time. To set a different application context for the session, you can:

- Call the CLEAR-APPL-CONTEXT( ) method, to clear the current application context, and then call the SET-APPL-CONTEXT( ) method with the new application context.
- Call the SET-APPL-CONTEXT( ) method with the new application context. If there is an existing application context in effect, Progress clears the existing application context before setting the new application context.

Calling this method does not generate an audit event or an audit record.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

See also APPL-CONTEXT-ID attribute, SET-APPL-CONTEXT( ) method

CLEAR-LOG( ) method

Clears all messages existing in the current client log file and leaves the file open for writing.

Note: This method is valid only for interactive and batch clients. WebSpeed agents and AppServers write a message to the server log file indicating that it is invalid to use the CLEAR-LOG( ) method to clear a WebSpeed or AppServer server log file. In this case, the method returns FALSE.

WebSpeed agents and AppServers silently ignore the Client Logging (`-clientlog`) startup parameter. The broker handles the clearing of the WebSpeed and AppServer server logs, through the `srvrLogAppend` property in the `ubroker.properties` file.

Return type: LOGICAL

Applies to: LOG-MANAGER system handle

Syntax

```
CLEAR-LOG( )
```

Notes

- If the CLEAR-LOG( ) method successfully clears the open log file, it returns TRUE.
- If the CLEAR-LOG( ) method fails, it returns FALSE and displays a warning message indicating the reason for the failure.

- If there is no client log file, the CLEAR-LOG( ) method returns FALSE and displays a warning message that the operation is not valid when there is no log file.
- If you specified a log file threshold with either the Log Threshold (`-logthreshold`) startup parameter or the LOG-THRESHOLD attribute of the LOG-MANAGER handle, the CLEAR-LOG( ) method deletes any existing log files that match the name of the LOGFILE-NAME attribute or Client Logging (`-clientlog`) startup parameter. The method then re-creates and opens the first log file in the sequence and changes the LOGFILE-NAME attribute to reflect this.

CLEAR-SELECTION( ) method

Removes the highlight from the currently selected text.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
CLEAR-SELECTION ( )
```

If the highlight is removed, the method returns TRUE. Otherwise, it returns FALSE.

CLIENT-CONNECTION-ID attribute

For a session-managed application, this attribute returns the connection ID for the physical AppServer connection associated with this server handle.

For a session-free application, the connection is a binding to an application service that relies on a pool of AppServer connections to service all requests from the client. This attribute returns the connection ID for the first physical AppServer connection associated with the server handle.

For Web services, this attribute returns the empty string.

Data type: CHARACTER

Access: Readable

Applies to: Server object handle

This value is assigned by the AppServer broker when an AppServer accepts a connection request from a client application. The AppServer broker and all AppServer agents use the connection ID as an identifier when they log any information associated with the connection.

The same connection ID is available to a 4GL client application using the CLIENT-CONNECTION-ID attribute and to the AppServer agent servicing the client on the same connection using the SERVER-CONNECTION-ID attribute on the SESSION handle.

The value of the connection ID is guaranteed to be globally unique for all time within a single computer network. Connection IDs can be compared to each other strictly for equality, but other types of comparisons are irrelevant.

For a client, the connection ID of the associated AppServer connection remains the same until the client disconnects from the AppServer. If the client reconnects to the same AppServer, the connection ID of the new connection (and thus the value of the CLIENT-CONNECTION-ID attribute for that connection) is different from the connection ID of the previous connection.

CLIENT-TTY attribute

Returns the name of the terminal display for this user's login session. If not specified, Progress returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the Client-principal object is sealed, this attribute is read-only.

See also CLIENT-TTY attribute

CLIENT-TYPE attribute

Returns the type of OpenEdge client currently executing.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

Table 55 shows the value of CLIENT-TYPE for each supported client type.

Table 55: CLIENT-TYPE attribute values

Type of client	Attribute value
ProVision standard 4GL client	4GLCLIENT
WebClient	WEBCLIENT
AppServer agent	APPSERVER
WebSpeed agent	WEBSPEED
Other special-purpose clients	Unknown value (?)

CLIENT-WORKSTATION attribute

The name of the host workstation on which the user, represented by the Client-principal object, is working. If not specified, Progress returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the Client-principal object is sealed, this attribute is read-only.

See also CLIENT-TTY attribute

CLONE-NODE( ) method

Clone the XML node referred to by a node reference. The first parameter must be a valid X-noderef handle and refers to the new cloned XML node if the method succeeds. The new node is associated with the same document, but needs to be inserted with INSERT-BEFORE( ) or APPEND-CHILD( ) to become part of the document structure.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
CLONE-NODE( x-node-handle , deep )
```

x-node-handle

A valid X-noderef handle to use for the new XML node.

deep

A logical that if TRUE specifies that the whole sub-tree is to be cloned. The default value is FALSE.

The following example demonstrates the use of the CLONE-NODE( ) method to clone an entire sub-tree:

```
hOldNode:CLONE-NODE(hNewNode,true).
```

CLOSE-LOG( ) method

Closes the current log file, which stops an interactive or batch client from writing messages to the log file.

The CLOSE-LOG( ) method writes a message to the log file indicating that the client intentionally closed the log file, so that the user knows why there are no more messages in the log file.

Note: This method is valid only for interactive and batch clients. WebSpeed agents and AppServer servers write a message to the server log file indicating that it is invalid to use the CLOSE-LOG( ) method to close a WebSpeed or AppServer server log file. In this case, the method returns FALSE.

Return type: LOGICAL

Applies to: LOG-MANAGER system handle

Syntax

```
CLOSE-LOG( )
```

Notes

- If the CLOSE-LOG( ) method successfully closes the open log file, it returns TRUE.
- If the CLOSE-LOG( ) method fails, it returns FALSE and displays a warning message indicating the reason for the failure.
- If there is no client log file, the CLOSE-LOG( ) method returns TRUE and does not display a warning message.

CODE attribute

A numeric code associated with the last event.

Data type: INTEGER

Access: Readable

Applies to: LAST-EVENT system handle

For keyboard and mouse events (EVENT-TYPE attribute set to "KEYPRESS" or "MOUSE"), this is the key code. For high-level Progress events (EVENT-TYPE attribute set to "PROGRESS"), this is a unique numeric value greater than the key code values. For information on key codes, see the chapter on handling user input in *OpenEdge Development: Programming Interfaces*.

If a mouse event is high-level mouse event (for example, MOUSE-SELECT-CLICK), this attribute is set to the key code of the low-level mouse event (for example, MOUSE-SELECT-UP) that triggered the high-level event. To determine the triggered high-level event, you must also check the value of the FUNCTION attribute, in this case "MOUSE-SELECT-CLICK".

CODEPAGE attribute

The code page of specified r-code.

Data type: CHARACTER

Access: Readable

Applies to: RCODE-INFO system handle

This attribute references the code page of the strings in the text segment. The code page value is written to the r-code file when the file is saved. Progress uses the code page specified by the R-code Out Code Page (-cprcodeout) startup parameter to write the r-code text segment. If -cprcodeout is not specified, Progress uses the value of the Internal Code Page (CPINTERNAL) SESSION handle.

For a file that is session compiled, the return value is the Unknown value (?).

COLUMN attribute

The column position of the left edge of the widget or the column position of the mouse cursor for the last mouse event on the display.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cells), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For browse cells, field groups, and the LAST-EVENT handle, it is readable only.

For all widgets except windows, the COLUMN attribute specifies the location, in character units, of the left edge of the widget relative to the left edge of its parent widget. In windows, the location is relative to the left edge of the display.

For browse columns, the COLUMN attribute returns the Unknown value (?) if the column is hidden.

For control-frames, the COLUMN attribute maps to the Left property of the control-frame COM object (ActiveX control container).

For the LAST-EVENT handle, the COLUMN attribute specifies the column location, in character units, of the last mouse event relative to the left edge of the current frame.

This attribute is functionally equivalent to the X attribute.

COLUMN-BGCOLOR attribute

The color number of the background color for the columns in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

COLUMN-DCOLOR attribute (Character interfaces only)

The number of the display color of a column.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

Overrides the color specified for the entire browse widget to display a single column in the specified color.

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

COLUMN-FGCOLOR attribute

The color number of the foreground color for the columns in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

COLUMN-FONT attribute (Graphical interfaces only)

The font for the columns in a browse widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget (column)

The font values are defined by your operating system.

COLUMN-LABEL attribute

A text string that describes a column of data associated with a buffer-field.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

COLUMN-MOVABLE attribute (Graphical interfaces only)

Indicates whether you can move a browse-column by pointing, clicking, and dragging.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget

If COLUMN-MOVABLE is TRUE, you can move any of the browse's browse-columns by pointing, clicking, and dragging on a column label. This attribute lets you turn on and off the ability of end users to move browse-columns. IF COLUMN-MOVABLE is TRUE, the START-MOVE and END-MOVE events for the browse-column take precedence over all other events. That is, while this attribute is TRUE, searching on columns is suspended, and Progress interprets the MOUSE-DOWN event as the column move.

The COLUMN-MOVABLE attribute has no effect on the MOVE-COLUMN( ) method, which lets you move a browse column programmatically.

For updateable browses, if a browse-column moves, its tab order changes.

When you set a browse's COLUMN-MOVABLE attribute to a certain value, Progress automatically sets the MOVABLE attribute of the browse's browse-columns to the same value. For more information on the MOVABLE attribute, see the MOVABLE attribute reference entry in this book.

COLUMN-PFCOLOR attribute (Character interfaces only)

The color number for the display color of a column with input focus.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

Overrides the color specified for the entire browse widget to display a single column in the specified color.

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

COLUMN-READ-ONLY attribute

Indicates whether you can tab to a browse-column but not edit it.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget (column)

Note: The COLUMN-READ-ONLY attribute of the browse-column corresponds to the READ-ONLY attribute of the fill-in, while the READ-ONLY attribute of the browse-column corresponds to the SENSITIVE attribute of the fill-in.

COLUMN-RESIZABLE attribute (Graphical interfaces only)

Indicates whether you can resize a browse-column by pointing, clicking, and dragging.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

If COLUMN-RESIZABLE is TRUE, you can resize any of the browse's browse-columns by pointing, clicking, and dragging on a column separator. This attribute lets you turn on and off the ability of end users to resize browse-columns. IF COLUMN-RESIZABLE is TRUE, the START-RESIZE and END-RESIZE events for the browse-column take precedence over all other events.

When you set a browse's COLUMN-RESIZABLE attribute to a certain value, Progress automatically sets the RESIZABLE attribute of the browse's browse-columns to the same value. For more information on the RESIZABLE attribute, see the RESIZABLE attribute reference entry in this book.

COLUMN-SCROLLING attribute (Windows only)

The horizontal scrolling behavior of a browse widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

If the COLUMN-SCROLLING attribute is TRUE, horizontal scrolling for the browse widget moves in whole-column increments. If a column is wider than the browse widget, you cannot see the text if the right side of the column, but you can still scroll to the next column.

If the attribute is FALSE, horizontal scrolling for the browse widget moves in increments equal to the pixel width of the average character in the current browse font. In this case, if a column is wider than the browse, you can view it by scrolling through the column in these increments.

The default value is TRUE for a static browse, and FALSE for a dynamic browse.

COM-HANDLE attribute (Windows only)

The component handle to the control-frame COM object.

Data type: COM-HANDLE

Access: Readable

Applies to: CONTROL-FRAME widget

This handle provides access to the ActiveX control container (COM object) associated with the control-frame. You can use this, in turn, to access control-frame COM object properties and methods.

For information on the properties and methods on the control-frame COM object, see the CONTROL-FRAME widget reference entry. For information on accessing COM object properties and methods, see the "Referencing COM object properties and methods" section on page 1501.

COMPLETE attribute

Indicates if the asynchronous request is completed and its result is processed on the client.

Data type: LOGICAL

Access: Readable

Applies to: Asynchronous request object handle

If set to FALSE, the PROCEDURE-COMPLETE event on this handle has not yet been returned from the AppServer running the request. This attribute is set to TRUE when the AppServer returns the PROCEDURE-COMPLETE event and immediately before any specified event procedure executes.

CONFIG-NAME( ) attribute

The WebSpeed service name. This method is called by the `get-config` WebSpeed API function. Intended for internal use only.

Data type: CHARACTER

Access: Readable

Applies to: WEB-CONTEXT system handle

CONNECT( ) method (AppServer)

Physically connects and associates an AppServer instance, or logically connects an application service, with the specified server handle. The current application becomes a client application of the connected AppServer.

Return type: LOGICAL

Applies to: Server object handle

Syntax

```
CONNECT ( [ connection-parms ] [ , userid ] [ , password ]
          [ , app-server-info ] )
```

All of the parameters for the CONNECT( ) method are optional and have defaults if you do not specify them.

connection-parms

A character string containing a space-separated list of one or more connection parameters necessary to establish an AppServer connection. These parameters include two types:

- A basic set used to connect to an AppServer instance or application service, regardless of the session model.
- A set for specifying and managing the session model of the connection.

Table 56 describes the basic connection parameters you must specify to connect to an AppServer instance or application service, regardless of the session model.

Table 56: AppServer basic connection parameters *(1 of 3)*

Connection parameter[1]	Description
`-AppService` *`application-service`*	If you connect through a NameServer, the name of an Application Service supported by the specified NameServer. (Defaults to the default service for the specified Name Server.) If you connect directly to an AppServer, this parameter is ignored.
`-H [` *`host_name`* `|` *`IP-address`* `]`	The network address to a NameServer machine or, if you connect directly, to an AppServer machine. You can specify either the TCP/IP host name or the Internet protocol address of the machine. (Defaults to `localhost`.)
`-S [` *`service-name`* `|` *`port-number`* `]`	The UDP port number for a NameServer, or, if you connect directly, the TCP/IP port number for an AppServer connection. You can specify either an explicit port number or a service name. If you use a service name, the method uses the port number associated with that name in the TCP/IP services file. (Defaults to `5162`.)
`-DirectConnect`	If specified, the `-H` and `-S` parameters are interpreted as the network address and TCP/IP port number of an AppServer connection. Otherwise, the `-H` and `-S` parameters are interpreted as the network address and UDP port number of a NameServer.

Table 56: AppServer basic connection parameters *(2 of 3)*

Connection parameter[1]	Description
`-ssl`	If specified, the connection is direct to the AppServer using Secure Sockets Layer (SSL) tunneling. (Used in conjunction with the `-AppService`, `-H`, and `-S` parameters). **Note:** Be sure you need SSL before using this option. SSL incurs more or less heavy performance penalties, depending on resources and load.
`-nosessionreuse`	If specified, the connection does not reuse the SSL session ID when reconnecting to the same SSL-enabled server (either a Web server with HTTPS or an SSL-enabled AppServer).
`-nohostverify`	If specified, turns off host verification for an SSL-enabled connection, either using HTTPS with the AIA or using a direct connection to an SSL-enabled AppServer. Without this parameter specified, the client compares the host name specified in the connection with the Common Name specified in the server certificate, and raises an error if they do not match. With this parameter specified, the client never raises the error. For more information, see *OpenEdge Getting Started: Core Business Services*.
`-pf` *filename*	A text file containing any of the other AppServer connection parameters described in this table or Table 57. If this file contains any other OpenEdge startup parameters, the method ignores them.

Table 56: AppServer basic connection parameters *(3 of 3)*

Connection parameter[1]	Description
`-URL` *`Web-or-AppServer-path`*	An HTTP (or HTTPS-based) URL to an AIA (for an Internet-secure AppServer connection) or an AppServer-based URL (with or without SSL tunneling for an SSL-enabled AppServer connection). For more information, see the sections on connecting to an AppServer using a URL in *OpenEdge Application Server: Developing AppServer Applications*. **Note:** Be sure you need SSL (either, and especially both, an HTTPS or SSL-enabled AppServer) before using this option. SSL at any point in a networked application incurs more or less heavy performance penalties, depending on resources and load.

[1] Previous versions of the AppServer allow you to include a network protocol using the `-N` parameter, which must always specify `TCP`. While still allowed, it is optional and always defaults to TCP/IP.

Note: Connections to an Internet-secure (HTTPS) or SSL-enabled AppServer require the management of public keys on the client (SSL client) and private keys on the server (SSL server). For an Internet-secure AppServer, the SSL server is the Web server that hosts the AIA. For an SSL-enabled AppServer, the SSL server is the AppServer itself. For information on configuring a Web server for HTTPS, see your Web server documentation. For information on using SSL to secure an AppServer, see *OpenEdge Application Server: Developing AppServer Applications*. For information on configuring an AppServer for SSL tunneling, see *OpenEdge Application Server: Administration*. For information on managing private key and digital certificate stores for SSL clients and servers, see *OpenEdge Getting Started: Core Business Services*.

Table 57 describes connection parameters for specifying and managing the session model of the connection.

Table 57: AppServer session model connection parameters *(1 of 3)*

Connection parameter	Session model/ default	Description
`-sessionModel` *`sessionModel`*	Session-managed Session-free	Session model supported by the AppServer operating mode, specified by one of the following values: • `Session-managed` • `Session-free` This value is **not** case sensitive. This parameter is required for session-free applications and is optional for session-managed applications. This value must match the AppServer operating mode or the `CONNECT( )` method fails. The default value is `Session-managed`.
`-connectionLifetime` *`nSeconds`*	Session-free	The maximum number of seconds that a given connection can be used before it is destroyed. Connections whose lifetime exceeds the specified value are destroyed as they become available. An available connection is one that is not currently reserved to run a request. Bound connections associated with remote persistent procedures are not available for re-use until the persistent procedure is deleted. So, bound connections remain available as long as necessary, even if they exceed the specified value. The default value is 300 seconds.

Table 57: AppServer session model connection parameters *(2 of 3)*

Connection parameter	Session model/ default	Description
`-initialConnections` *nConnections*	Session-free	The number of connections established when the CONNECT( ) method executes on a given server handle. The value must be greater than zero. If the specified number of connections cannot be created, the CONNECT( ) method fails and any successfully-created connections are closed. The default value is 1.
`-maxConnections` *nConnections*	Session-free	The maximum number of connections that can be created for a given server handle to execute non-persistent external procedures. The value must be greater than or equal to zero. If this value is zero, there is no limit to the number of connections that can be created. **Note:** For calls to persistent procedures, their internal procedures, and user-defined functions, the client has no limit on the number of connections that can be created. The default value is 0.
`-nsClientMaxPort` *portNum*	Session-manage Session-free	The maximum value for the UDP port number used by the client when communicating with the NameServer. If this value is zero, OpenEdge chooses the NameServer client port randomly. This value should be greater than or equal to the value of the `-nsClientMinPort` parameter. The default value is 0.

Table 57: AppServer session model connection parameters *(3 of 3)*

Connection parameter	Session model/ default	Description
-nsClientMinPort *portNum*	Session-manage Session-free	The minimum value for the UDP port number used by the client when communicating with the NameServer. If this value is zero, OpenEdge chooses the NameServer client port randomly. The default value is 0.
-nsClientPicklistExpiration *nSeconds*	Session-free	The maximum amount of time, in seconds, that the client retains an AppServer pick list for an application service. The default value is 300.
-nsClientPicklistSize *nPicks*	Session-free	The number of AppServer picks to request from the NameServer each time it looks up the available AppServer connections for a given application service name. The default value is 1.
-nsClientPortRetry *nRetries*	Session-manage Session-free	The maximum number of attempts that the client makes to get a valid local UDP port number when attempting to communicate with the NameServer. The default value is 0.
-nsClientDelay *nMilliSeconds*	Session-manage Session-free	The interval, in milliseconds, that the client waits between attempts to get a valid UDP port number when attempting to communicate with the NameServer. The default value is 0.

Note that the actual AppServer that the client connects to is controlled by the NameServer based on the application service (`-AppService`) name specified by the client. The OpenEdge interface in cooperation with the NameServer connect the client application to one of the AppServer instances that supports the specified application service. If you do not specify an application service, the NameServer uses whatever AppServer registers itself as the default service, if any. For more information on load balancing, see the information on NameServers and load balancing in *OpenEdge Application Server: Developing AppServer Applications* and the AppServer administration chapter in *OpenEdge Application Server: Administration*.

If the application service is unknown to the NameServer, the client application receives an error. Otherwise, the connection proceeds and any configured Connect procedure executes for the connected AppServer.

For more information on application services and NameServers, see *OpenEdge Application Server: Developing AppServer Applications*

[*userid* **]** **[** , *password* **]** **[** , *app-server-info* **]**

From one to three character string parameters passed as input to the AppServer Connect procedure. The possible values that you can specify for these parameters is determined by the Connect procedure for the AppServer application. If you omit a parameter, it defaults to the Unknown value (`?`).

If an error occurs while executing the CONNECT( ) method, the method returns FALSE. Otherwise, it returns TRUE. An error can occur if:

- The server handle is invalid.
- One of the parameters contains an invalid value.
- One of the values specified in the *connection-parms* parameter is invalid.
- The Name Server cannot be located.
- The specified Application Service is not registered to a NameServer.

- The client application cannot connect to the AppServer selected by the NameServer.
- The AppServer selected by the NameServer cannot allocate a connection for the client application.
- The AppServer executes a Connect procedure that terminates with a STOP condition, a QUIT condition, or after executing a RETURN ERROR statement. For more information on Connect procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

If the CONNECT( ) method completes successfully, the CONNECTED( ) method returns TRUE.

The connection lasts until the client application executes the server handle DISCONNECT( ) method or until Progress detects any failure conditions that automatically terminate the connection.

The `-URL` connection parameter allows you to connect to an AppServer using the AppServer Internet Adapter (AIA) with the following protocols: HTTP and HTTPS.

For more information on AppServers or the AppServer Internet Adapter (AIA), see *OpenEdge Application Server: Developing AppServer Applications*.

CONNECT( ) method (Socket object)

Connects a socket to the specified TCP/IP port on the specified host.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
CONNECT ( [ connection-parms ] )
```

connection-parms

A character string expression that contains a space-separated list of one or more socket connection parameters.

Table 58 describes each socket connection parameter, which can be included in this string.

Table 58: Socket connection parameters *(1 of 2)*

Parameter	Description
-H *socket-address*	Optional. The host name or IP address to which the connection is to be established.
-S *socket-port*	The port number for the socket connection. You can specify either an explicit port number or a TCP service name. If you use a TCP service name, the method uses the port number associated with that name in the TCP/IP services file.
-ssl	If specified, the connection to the server socket uses Secure Sockets Layer (SSL) tunneling. (Used in conjunction with the -H and -S parameters). **Note:** Be sure you need SSL before using this option. SSL incurs more or less heavy performance penalties, depending on resources and load.
-nosessionreuse	If specified, the connection does not reuse the SSL session ID when reconnecting to the same SSL-enabled server socket.

Table 58: Socket connection parameters *(2 of 2)*

Parameter	Description
`-nohostverify`	If specified, turns off host verification for an SSL-enabled connection to a server socket. Without this parameter specified, the client compares the host name specified in the connection with the Common Name specified in the server certificate, and raises an error if they do not match. With this parameter specified, the client never raises the error. For more information, see *OpenEdge Getting Started: Core Business Services*.
-pf *filename*	Optional. A text file containing any of the socket connection parameters described in this table. If this file contains any other OpenEdge startup parameters, this method ignores them.

Note: Connections to an SSL-enabled server socket require the management of public keys on the client (SSL client) and private keys on the server (SSL server). For 4GL sockets, the SSL client is the 4GL session initiating the SSL connection on a socket object and the SSL server is the 4GL session enabling SSL connections on a server socket object. For information on using SSL to secure a 4GL socket connection, see the sections on sockets in *OpenEdge Development: Programming Interfaces*. For more information on SSL and managing private key and digital certificate stores for OpenEdge SSL clients and servers, see *OpenEdge Getting Started: Core Business Services*.

Notes

- If an error occurs while executing the CONNECT( ) method, the method returns FALSE. Otherwise, it returns TRUE.
- When a 4GL client (that is not SSL-enabled) calls the CONNECT( ) method and immediately reads data from the socket using the READ( ) method, and a 4GL server (that is SSL-enabled) calls the ENABLE-CONNECTIONS( ) method and immediately writes data to the socket using the WRITE( ) method, a deadlock condition can occur. That is, the client is waiting for the server to send data, and the server (regardless of the Write operation) is waiting for the client connection to send data that starts the SSL connection.

CONNECT( ) method (Web service)

Connects to and associates a Web service instance with the specified server handle. The current application becomes a client application of the connected Web service.

Return type: LOGICAL

Applies to: Server object handle

Syntax

```
CONNECT ( [ connection-parms ] )
```

connection-parms

A character string containing a space-separated list of one or more connection parameters.

Table 59 describes each Web service connection parameter you can include in this string. This method ignores any other strings included in this parameter.

Table 59: Web service connection parameters *(1 of 4)*

Parameter	Description
-WSDL *wsdl-document*	The location of the WSDL document. This required parameter is the URL, UNC, or local file pathname to the WSDL file that describes the Web service. The document can be local or remote. The location can optionally contain a user's account name and password to use when connecting to the Web Server. If the protocol is not part of the wsdl-document's URL, the 'file' protocol is assumed. Additionally, the 'file' can be a relative pathname as it is relative to the current working directory.
-WSDLUserid *user-id*	Optional user account name to use in connecting to the Web services Adapter (WSA) that hosts the WSDL document. If -WSDLUserid is specified and -WSDLPassword is not, OpenEdge uses a blank password.

Table 59: Web service connection parameters *(2 of 4)*

Parameter	Description
`-WSDLPassword` *password*	Optional password to use with `-WSDLUserid`. This attribute is ignored if `-WSDLUserid` is not specified.
`-Service` *service-name*	The local name of the service element within the WSDL document that the application will use. This field is optional. Many WSDL documents only support one service and this parameter is optional if there is only one (or zero) service elements defined. Used in conjunction with `-Port`.
`-ServiceNamespace` *service-namespace*	The namespace of the service element within the WSDL document that the application will use. Most WSDL documents only support one service and this parameter is optional if there is only one service defined. This parameter is used in conjunction with `-Service` and is ignored if `-Service` was not specified. This parameter is optional, if the namespace is included in `-Service`.
`-Port` *port-name*	The local name of the port element contained within the service element. Used in conjunction with `-Service`. This parameter is optional if `-Service` contains only one port.
`-Binding` *binding-name*	The local name of the binding element contained in the WSDL document. Used in conjunction with `-SoapEndpoint`. This parameter is optional if the WSDL contains only one binding.
`-BindingNamespace` *binding-namespace*	The namespace of the binding element within the WSDL document that the application will use. This optional field is needed only if the local binding-name is not unique.
`-SOAPEndpoint` *URL-endpoint*	The URL identifying the endpoint for this Web service. Used in conjunction with `-Binding`. It is an error to use this parameter in conjunction with `-Service` or `-Port`.

Table 59: Web service connection parameters *(3 of 4)*

Parameter	Description
-SOAPEndpointUserid *user-id*	Optional user account name to use to connect to a Web service that hosts the Web Server application. If -SOAPEndpointUserid is specified and -SOAPEndpointPassword is not, OpenEdge uses a blank password.
-SOAPEndpointPassword *password*	Optional password to use with the -SoapEndpointUserid. This attribute is ignored if -SoapEndpointUserid is not specified.
-TargetNamespace *targetNamespace*	The namespace contained in the WSDL document. This parameter can be used as a version check. The information in this parameter is compared against the -TargetNamespace contained in the WSDL document. If they do NOT match the CONNECT ( ) method fails.
-connectionLifetime *nSeconds*	The maximum number of seconds that a given connection can be reused for asynchronous requests before it is destroyed. Connections whose lifetime exceeds the specified value are destroyed as they become available. An available connection is one that is not currently reserved to run an asynchronous request. The default value is 300 seconds.
-maxConnections *num-connections*	Maximum number of connections maintained between the client and the Web Server for asynchronous requests. If num connections is less than or equal to 0, the application is requesting no predefined limit on the number of connections. If the client application exceeds the specified number of connections, the asynchronous requests are queued. The default value is 0.
-nosessionreuse	If specified, the connection does not reuse the SSL session ID when reconnecting to the same HTTPS-enabled Web server.

Table 59: Web service connection parameters *(4 of 4)*

Parameter	Description
`-nohostverify`	If specified, turns off host verification for an HTTPS Web server connection. Without this parameter specified, the client compares the host name specified in the connection with the Common Name specified in the server certificate, and raises an error if they do not match. With this parameter specified, the client never raises the error. For more information, see *OpenEdge Getting Started: Core Business Services*.
`-pf` *`filename`*	A text file containing any of the other Web service binding parameters described in this table. If this file contains any other OpenEdge startup parameters, this method ignores them.

Note: Connections to an Internet-secure (HTTPS) Web service require the management of public keys on the client (HTTPS client) and private keys on the server (HTTPS server). For an Internet-secure Web service, the HTTPS server is the Web server that hosts the Web service. For information on configuring a Web server for HTTPS, see your Web server documentation. For more information on HTTPS and SSL, and on managing private key and digital certificate stores for OpenEdge SSL clients and servers, see *OpenEdge Getting Started: Core Business Services*.

The WSDL parameter can optionally contain a user account name and password to use to connect to a Web Server. The syntax for the HTTP and HTTPS protocol is:

```
-WSDL
http://[user-id[:password]@]web-server-host[:web-server-port]WSDL-path
```

```
-WSDL
https://[user-id[:password]@]web-server-host[:web-server-port]WSDL-path
```

user-id

User account name to use to connect to a Web service that hosts the WSDL document. If user-id is specified and password is not, OpenEdge uses a blank password.

password

Password to use with the user-id. This parameter is ignored if user-id is not specified.

web-server-host

TCP/IP host address of the Web Server that hosts the WSDL document.

web-server-port

TCP/IP port address of the Web Server that host the WSDL document. The default port is 80 for HTTP and 443 for HTTPS.

WSDL-path

URL path to the WSDL document for the Web service.

Instead of building the account name and password into the WSDL string, you can specify the account name via the `-WSDLUserid` parameter and the password via the `-WSDLPassword` parameter. If these parameters are used and the WSDL URL also contains a user id and password, the information on the WSDL URL is used.

Connection parameter combinations

The CONNECT( ) method is used to connect a Progress SERVER object to a specific application service. This service can be either an AppServer or a Web service. Independent of the type of application service to which the client is connecting, the client needs to provide the location of the service and transport information. There are two mechanisms for providing this information when connecting to a Web service:

1. The CONNECT( ) method can identify a specific service element name and port element name from the WSDL document. The combination of these two element names identify the location of a set of operations that are available on the Web service. It also identifies the transport data. The service element name is specified with the `-Service` connection parameter and the port element name is specified with the `-Port` connection parameter.

   If the WSDL document contains several service elements, the CONNECT method must identify which service element the client wants to connect to, via `-Service`. If the WSDL document only identifies one service element, the CONNECT method does not need to contain the service element name. Similarly if the WSDL document (or if the identified service element) only identifies one port element, the CONNECT method does not need to contain the port element name.

   If the application needs to provide account name and password information, it can accomplish this by providing the account name and password information in the `-SoapEndpointUserid` and `-SoapEndpointPassword` parameters.

   If the WSDL document identifies multiple service elements with the same local name, the CONNECT ( ) method must also contain the `-ServiceNamespace` connection parameter.

2. Some WSDL documents do not contain a service element. The client application can provide the same logical information by providing the binding element name. The binding element name identifies the transport data and a set of operations that are available on the Web service, but it does not identify the location of this Web service. Therefore, when specifying the `-Binding` parameter, the CONNECT method must also contain the `-SoapEndpoint` parameter to identify the location of the Web service.

   If the WSDL document contains several binding elements, the CONNECT method must identify which binding element the client wants to use, via the `-Binding` parameter. If the WSDL document only identifies one binding element, the CONNECT method does not need to contain the binding element name.

If the application needs to provide account name and password information, it can accomplish this by providing the account name and password information in the `-SoapEndpointUserid` and `-SoapEndpointPassword` parameters.

If the WSDL document identifies multiple binding elements with the same local name, the CONNECT( ) method must also contain the `-BindingNamespace` connection parameter.

If an error occurs while executing the CONNECT( ) method, the method returns FALSE. Otherwise, it returns TRUE. An error can occur if:

- The server handle is invalid.
- One of the parameters contains an invalid value.
- One of the values specified in the *`connection-parms`* parameter is invalid.
- The `-TargetNamespace` does not match the value contained in the WSDL document.
- The WSDL document cannot be located.
- The `-WSDLUserid` or `-WSDLPassword` is not valid.

If the CONNECT( ) method completes successfully, the CONNECTED( ) method returns TRUE.

The connection lasts until the client application executes the server handle DISCONNECT( ) method or until Progress detects any failure condition that automatically terminates the connection.

CONNECTED( ) method

Indicates whether an AppServer or Web service is currently connected and associated with the server handle, or if a socket handle is currently connected to a port.

Note: For a Web service, this method indicates if a server handle is currently connected to a Web service (that is, if the client has a logical connection to the Web service). It does not indicate that a physical connection exists between the OpenEdge client and the Web service.

Return type: LOGICAL

Applies to: Server object handle, Socket object handle

Syntax

```
CONNECTED ( )
```

For a state-reset, state-aware, or stateless AppServer, this method returns TRUE if the AppServer is currently connected and associated with the server handle. For a state-free AppServer, this method returns TRUE if the CONNECT( ) method has been successfully executed for an application service associated with this handle and at least one AppServer resource is available for the client to access this application service.

For a Web service, this method returns TRUE if the server handle refers to a connected Web service, and returns FALSE otherwise.

This method returns TRUE between the successful invocation of the CONNECT( ) method and a call to the DISCONNECT( ) method. If a server handle was connected to an AppServer or Web service, but the connection terminated abnormally (that is, other than by the DISCONNECT( ) method), the CONNECTED( ) method returns FALSE.

Note: This method returning TRUE does not indicate the state of the HTTP connection to the Web service. If there is a failure in the connection between the client and the Web service, subsequent requests might fail.

For a socket object, this method returns TRUE if the socket handle refers to a connected socket, and returns FALSE otherwise.

For more information on AppServers, see *OpenEdge Application Server: Developing AppServer Applications*.

CONTEXT-HELP attribute (Windows only)

When CONTEXT-HELP is TRUE, a question mark icon displays in the title bar of the window or dialog box. The default value is FALSE. This attribute must be set before the window or dialog box is realized.

Return type: LOGICAL

Access: Readable/Writable

Applies to: DIALOG-BOX widget, WINDOW widget

Due to bugs in Microsoft Windows, the question mark icon does not appear, or appears but does not function, when combined with other attribute settings that affect a window's title bar:

- If CONTEXT-HELP = TRUE and SMALL-TITLE = TRUE, the question mark icon does not appear.
- If CONTEXT-HELP = TRUE and both MIN-BUTTON = TRUE and MAX-BUTTON = TRUE, the question mark icon does not appear.
- If CONTEXT-HELP = TRUE and either (but not both) of MIN-BUTTON or MAX-BUTTON = TRUE, the question mark icon appears but does not function.
- If CONTEXT-HELP = TRUE and CONTROL-BOX = FALSE, the question mark icon does not appear.

To summarize, you must set CONTEXT-HELP = TRUE, MIN-BUTTON = FALSE, and MAX-BUTTON = FALSE (leaving CONTROL-BOX at its default value of TRUE and SMALL-TITLE at its default value of FALSE) in order to successfully use this feature with a window widget.

Note: The preceding settings only apply to window widgets, not to dialog boxes. The question mark icon always functions correctly when used with a dialog box.

CONTEXT-HELP-FILE attribute (Windows only)

Specifies the path name of a help (`.HLP`) file associated with a dialog box, window, or session.

Return type: CHARACTER

Access: Readable/Writable

Applies to: SESSION system handle, DIALOG-BOX widget, WINDOW widget

If CONTEXT-HELP-FILE is not specified (is unknown) for a dialog box, the dialog box inherits the help file of its parent window. If the parent window's CONTEXT-HELP-FILE is also unknown, it inherits the session's help file (specified by SESSION:CONTEXT-HELP-FILE). The full pathname of the help file should be given. Progress does not search for the help file.

CONTEXT-HELP-ID attribute (Windows only)

Specifies the identifier of a help topic in a help file.

Return type: INTEGER

Access: Readable/Writable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

CONTROL-BOX attribute (Windows only)

Indicates whether the window has a system menu box in its caption bar.

Data type: LOGICAL

Access: Readable/Writable

Applies to: WINDOW widget

In character interfaces, this attribute has no effect.

The CONTROL-BOX attribute must be set before the window is realized. The default value is TRUE.

Control-Name property (Windows only; Graphical interfaces only)

The component handle to an ActiveX control that has the specified design-time name (*Control-Name*) and that is loaded into the control-frame.

Data type: COM-HANDLE

Access: Readable

Applies to: CONTROL-FRAME widget, COM object

To return the component handle of the ActiveX control, you provide the design-time name as a property of the control-frame COM object:

```
DEFINE VARIABLE hSpin AS COM-HANDLE.
DEFINE VARIABLE hControlFrameCOM AS COM-HANDLE.
DEFINE VARIABLE hControlFrame AS WIDGET-HANDLE.

/* ... Instantiate the control-frame with hControlFrame
   and load the Spin control into it ... */

hControlFrameCOM = hControlFrame:COM-HANDLE.
hSpin = hControlFrameCOM:Spin.
```

In this example, Spin is the name of an ActiveX control and is also used as a property to return the handle to that control.

This is the simplest technique to access an ActiveX control that is loaded in a control frame.

Notes

- Another way of getting a component handle to an ActiveX control is to access it through the control collection. For more information, see the Controls property entry.
- References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.

Controls property (Windows only; Graphical interfaces only)

The component handle to the control collection that references the ActiveX controls in the control-frame.

Data type: COM-HANDLE

Access: Readable

Applies to: CONTROL-FRAME widget, COM object

Because this release supports only one ActiveX control per control-frame, this control collection references only one control. Once you have the component handle to the control collection (*collection-handle*), you can get the component handle to the ActiveX control itself by invoking the Item( ) method of the control collection. (This is a standard ActiveX convention.) With support limited to a single control per control-frame, the only valid Item( ) method call is *collection-handle*:Item(1). Once you have the component handle to the actual ActiveX control, you can access the properties and methods of that control.

Notes

- A simpler technique for getting a component handle to an ActiveX control is to reference its name directly as a property of the control-frame COM object. For more information, see the Control-Name property reference entry.
- References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.

CONVERT-3D-COLORS attribute (Windows only)

Determines whether image colors are converted to the corresponding system 3D colors.

Return type: LOGICAL

Access: Readable/Writable

Applies to: BUTTON widget, IMAGE widget

The default value of this attribute for a button is TRUE; the default value for an image is FALSE.

The CONVERT-3D-COLORS attribute can be set after a widget is realized, but it will not take effect until an image is loaded into the widget using one of the following methods: LOAD-IMAGE( ), LOAD-IMAGE-UP( ), LOAD-IMAGE-DOWN( ), or LOAD-IMAGE-INSENSITIVE( ).

If the CONVERT-3D-COLORS attribute is TRUE either when the widget is realized or when any of the LOAD-IMAGE methods is executed, Progress will convert the shades of gray in the image after loading it. Table 60 identifies and describes the colors that are converted:

Table 60: 3D-color conversions

If the color is...	And the original Red-Green-Blue (RGB) color value is...	Then the new converted system color is...
White	(255, 255, 255)	System button highlight color.
Light Gray	(192, 192, 192)	System button face color.
Dark Gray	(128, 128, 128)	System button shadow color.
Black	(0, 0, 0)	System button text color.

CONVERT-TO-OFFSET( ) method

Converts a row and column value to a character offset in an editor widget.

Return type: INTEGER

Applies to: EDITOR widget

Syntax

```
CONVERT-TO-OFFSET ( row , column )
```

row

An integer row number.

column

An integer column number.

In Windows, both the regular editor and the large editor support CONVERT-TO-OFFSET.

COPY-DATASET( ) method

Copies a source ProDataSet object to a target ProDataSet object. Progress empties the target ProDataSet object temp-tables of all records before copying the source ProDataSet object, by default.

Return type: LOGICAL

Applies to: ProDataSet object handle

Syntax

```
targ-dataset-handle:COPY-DATASET( src-dataset-handle [, append-mode
[, replace-mode [, loose-copy-mode [, pairs-list [, current-only ]
] ] ] ] )
```

targ-dataset-handle

The handle to the target ProDataSet object to receive the copy.

src-dataset-handle

The handle to the source ProDataSet object from which to copy.

append-mode

An optional logical expression where TRUE indicates that Progress copy the ProDataSet object temp-tables in an *append* mode.

When TRUE, Progress appends the source ProDataSet temp-tables to the target ProDataSet temp-tables. If there is a unique index on a target temp-table and Progress finds a row with a duplicate key, it does not replace the row. If there is not a unique index on the target temp-table, Progress appends the data row from the source temp-table to the target temp-table, which might result in duplicate rows. If this operation results in duplicate rows, Progress does not generate a run-time error.

Note: In this mode, Progress treats duplicate temp-table rows during the copy operation in the same way it treats duplicate temp-table rows during a fill operation in MERGE mode.

When you are certain the ProDataSet temp-tables do not contain duplicate rows, copying the ProDataSet object in append mode is more efficient than copying it in replace mode.

When FALSE, Progress does not append records in the target ProDataSet temp-tables. The default value is FALSE.

Progress ignores this expression when *replace-mode* is TRUE.

replace-mode

An optional logical expression where TRUE indicates that Progress copy the ProDataSet object temp-tables in a *replace* mode.

When TRUE, Progress replaces records in the target ProDataSet object temp-tables with corresponding records from the source ProDataSet temp-tables. In this case, the source and target temp-tables must be named differently, the target temp-table must have a unique primary index that Progress can use to find the corresponding records. When a corresponding record is found in the target temp-table, Progress replaces the target record with the source record. When a corresponding record is not found in the target temp-table, Progress creates a new target record using the source record. If the before-image table associated with the target temp-table contains a row for the target record, the row is left in place.

Copying ProDataSet object temp-tables in replace mode is less efficient than copying them in append mode. When you are certain the ProDataSet object temp-tables do not contain duplicate rows, copy the ProDataSet object in append mode.

When FALSE, Progress does not replace records in the target ProDataSet object temp-tables. The default value is FALSE.

loose-copy-mode

An optional logical expression where TRUE indicates that Progress copy the ProDataSet object temp-tables in a *loose-copy* mode. That is, it relaxes the requirement that the meta-schema for the source and target temp-tables be the same.

When TRUE, Progress copies each temp-table in the source ProDataSet object to the target ProDataSet object based on a field mapping between the source and target temp-table buffers. If there is an attached data source with a field mapping, Progress uses that field mapping to copy fields from each source temp-table buffer to its target temp-table buffer. If there are fields in either buffer that do not exist in the other, they are ignored. If there is no field mapping with the attached data source, or there is no attached data source, Progress copies only those fields that appear in both the source and target temp-table meta-schemas with the same name.

When FALSE, the meta-schema for the source and target temp-tables must be the same or Progress generates a run-time error. The default value is FALSE.

pairs-list

An optional character expression that evaluates to a comma-separated list of the target and source temp-table pairs to be copied. Following is the *pairs-list* syntax:

```
target-table1, source-table1 [, target-table2, source-table2 ] ...)
```

If specified, Progress copies only the listed temp-tables by matching the target and source temp-table names.

If not specified, Progress copies all the temp-tables in the order they were defined or added in the ProDataSets. If either the source or target ProDataSet has one or more extra temp-tables at the end, the extra temp-tables are ignored.

current-only

An optional logical expression where TRUE indicates that Progress copy only the current record from each temp-table at each level in the source ProDataSet object to the target ProDataSet object. The default value is FALSE.

Note: You might need to synchronize the buffers to ensure they are the related buffers. Once you have read a record into a top-level buffer, you can synchronize the related buffers by calling the SYNCHRONIZE( ) method.

To copy the current record from a single temp-table, you can use the BUFFER-COPY statement or BUFFER-COPY( ) method.

Notes

- If the source ProDataSet object has any before-image tables that contain changed row data for any associated temp-tables, Progress generates a run-time error.
- When Progress copies the source ProDataSet object, it copies each temp-table in the order in which the temp-table was defined, and in its entirety. That is, it **does not** copy the temp-tables in an interleaved and nested manner based on their data relations. If either the source or target ProDataSet has one or more extra temp-tables at the end, the extra temp-tables are ignored.
- When Progress copies a ProDataSet object in any mode, except loose-copy mode, and the target ProDataSet object has a meta-schema (that is, temp-table and relation definitions), the source ProDataSet object meta-schema must be the same. If the source and target ProDataSet object meta-schema is not the same, Progress generates a run-time error. If the target ProDataSet object is a newly created dynamic object with only a handle and no meta-schema, Progress copies the source ProDataSet object including its meta-schema. In this case, Progress names the temp-tables in the target object by taking the names of the temp-tables in the source object and prepending a "cpy_" prefix to them.

See also ATTACH-DATA-SOURCE( ) method, FILL-MODE attribute, COPY-TEMP-TABLE( ) method

COPY-TEMP-TABLE( ) method

Copies a source temp-table object to a target temp-table object. Either of the temp-tables (source or target) may be a member of a ProDataSet object. Progress empties the target temp-table of all records before copying the source temp-table, by default.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
targ-tt-handle:COPY-TEMP-TABLE( src-tt-handle [, append-mode
[, replace-mode [, loose-copy-mode ] ] ] )
```

targ-tt-handle

The handle to the target temp-table object to receive the copy.

src-tt-handle

The handle to the source temp-table object from which to copy.

append-mode

An optional logical expression where TRUE indicates that Progress copy the temp-table object in an *append* mode.

When TRUE, Progress appends the source temp-table object to the target temp-table object. If there is a unique index on the target temp-table and Progress finds a row with a duplicate key, it does not replace the row. If there is not a unique index on the target temp-table, Progress appends the data row from the source temp-table to the target temp-table, which might result in duplicate rows. If this operation results in duplicate rows, Progress does not generate a run-time error.

Note: In this mode, Progress treats duplicate temp-table rows during the copy operation in the same way it treats duplicate temp-table rows during a fill operation in MERGE mode.

When you are certain a temp-table object does not contain duplicate rows, copying the object in append mode is more efficient than copying it in replace mode.

When FALSE, Progress does not append records in the target temp-table object. The default value is FALSE.

Progress ignores this expression when *replace-mode* is TRUE.

replace-mode

An optional logical expression where TRUE indicates that Progress copy the temp-table object in a *replace* mode.

When TRUE, Progress replaces records in the target temp-table object with corresponding records from the source temp-table object. In this case, the source and target temp-tables must be named differently, and the target temp-table must have a unique primary index that Progress can use to find the corresponding record. When the corresponding record is found in the target temp-table, Progress replaces the target record with the source record. When the corresponding record is not found in the target temp-table, Progress creates a new target record using the source record. If the before-image table associated with the target temp-table contains a row for the target record, the row is left in place.

Copying a temp-table object in replace mode is less efficient than copying it in append mode. When you are certain a temp-table object does not contain duplicate rows, copy the object in append mode.

When FALSE, Progress does not replace records in the target temp-table object. The default value is FALSE.

loose-copy-mode

An optional logical expression where TRUE indicates that Progress copy the temp-table object in a *loose-copy* mode. That is, it relaxes the requirement that the meta-schema for the source and target temp-tables be the same.

When TRUE, Progress copies the source temp-table object to the target temp-table object based on a field mapping between the source and target temp-table buffers. If there is an attached data source with a field mapping, Progress uses that field mapping to copy fields from the source temp-table buffer to its target temp-table buffer. If there are fields in either buffer that do not exist in the other, they are ignored. If there is no field mapping with the attached data source, or there is no attached data source, Progress copies only those fields that appear in both the source and target temp-table meta-schema with the same name.

When FALSE, the meta-schema for the source and target temp-tables must be the same or Progress generates a run-time error. The default value is FALSE.

Notes

- If the source temp-table object has a before-image table that contains changed row data, Progress generates a run-time error.
- When Progress copies a temp-table object in any mode, except loose-copy mode, and the target temp-table object is in a PREPARED state (that is, it has a meta-schema), the source temp-table object meta-schema must be the same. Each column in the source temp-table must match the target temp-table in position, data type, and extent. If the source temp-table object meta-schema is not the same, Progress generates a run-time error. If the target temp-table object is not in a PREPARED state (that is, it has no meta-schema), Progress copies the source temp-table object including its meta-schema. In this case, Progress names the target temp-table by taking the name of the source temp-table and prepending a "cpy_" prefix to it.
- If the target temp-table object is a member of a ProDataSet object, Progress does not track changes to the data in that temp-table (it ignores the TRACKING-CHANGES attribute setting during the copy operation).

See also FILL-MODE attribute, COPY-DATASET( ) method

CPCASE attribute

The case table Progress uses to establish case rules for the Internal Code Page (-cpinternal) startup parameter.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute reads the value you set using the Case Table (-cpcase) startup parameter.

CPCOLL attribute

The collation table Progress uses with the Internal Code Page (`-cpinternal`) startup parameter.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

By default, Progress uses the collation rules you specify to compare characters and sort records. The collation rules specified with the Collation Table (`-cpcoll`) startup parameter take precedence over a collation specified for any database Progress accesses during the session, except when Progress uses or modifies pre-existing indexes. If you do not specify a collation with the `-cpcoll` startup parameter, Progress uses the language collation rules defined for the first database on the command line. If you do not specify a database on the command line, Progress uses the collation rules with the default name "`basic`" (which might or might not exist in the `convmap.cp` file).

CPINTERNAL attribute

The internal code page Progress uses in memory.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute reads the value you set using the Internal Code Page (-cpinternal) startup parameter.

CPLOG attribute

The code page for all messages written to the log (`.lg`) file.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute reads the value you set using the Log File Code Page (-cplog) startup parameter.

CPPRINT attribute

The code page Progress uses for the OUTPUT TO PRINTER statement.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute reads the value you set using the Printer Code Page (-cpprint) startup parameter.

CPRCODEIN attribute

The code page Progress uses to convert text strings into the text segment.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute reads the value you set using the R-code In Code Page (-cprcodein) startup parameter.

CPRCODEOUT attribute

The code page Progress uses at compile time to convert text strings into the text segment and marks the text segment with the code page name.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute reads the value you set using the R-code Out Code Page (-cprcodeout) startup parameter.

CPSTREAM attribute

The code page Progress uses for stream I/O.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute reads the value you set using the Stream Code Page (-cpstream) startup parameter.

CPTERM attribute

The code page Progress uses for I/O with character terminals.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute reads the value you set using the Terminal Code Page (-cpterm) startup parameter.

CRC-VALUE attribute

The cyclic redundancy check (CRC) value for either an r-code file, or a database table corresponding to a buffer object.

Data type: INTEGER

Access: Readable

Applies to: Buffer object handle, RCODE-INFO system handle

When applied to the RCODE-INFO system handle, the r-code CRC is calculated using the filename and contents of the r-code file specified by the RCODE-INFO:FILE-NAME attribute.

When applied to the Buffer object handle, the database CRC is calculated using the metaschema _CRC field value from the _File record for the database record corresponding to the buffer object's table (which can be a standard or temporary table).

The CRC for a temporary table is calculated differently than for a standard table. Some differences include:

- Standard tables have a _File record in the database that contains a _CRC field for the CRC value, which is calculated as you make changes to the table. Temporary tables do not have a _File record, and do not exist in a database.
- The CRC values for both standard and temporary tables include the datatype, extent and position of each column in the table, as well as index information. However, the CRC value for a standard table includes additional information that a CRC value for a temporary table does not (such as, the _Order field in the _File record).
- The CRC value for a standard table is stored in a .r file. Progress uses that CRC value at runtime to verify the integrity of application r-code that uses the table. Progress uses the CRC value for a temporary table to compare table parameters between a calling and called procedure (to avoid a field-by-field comparison).

For more information on CRCs, see *OpenEdge Deployment: Managing 4GL Applications*.

CREATE-LIKE( ) method

Creates a table like another existing table, or a dynamic ProDataSet object like another static or dynamic ProDataSet object.

Return type: LOGICAL

Applies to: ProDataSet object handle, Temp-table object handle

Syntax

```
CREATE-LIKE( { src-table-handle-exp | src-table-name-exp }
   [ , src-index-name-exp ] )
```

```
CREATE-LIKE( { src-dataset-handle | src-dataset-name }
   [ , name-prefix ] )
```

src-table-handle-exp

An expression that evaluates to a table handle from which to copy the field definitions, and optionally, the indexes if *src-index-name-exp* is not specified.

src-table-name-exp

An expression that evaluates to a table name from which to copy the field definitions and, optionally, the indexes if *src-index-name-exp* is not specified.

src-index-name-exp

A character expression giving an index to be copied from the source table. If this option is specified, only this single index is copied from the source table.

src-dataset-handle

The handle to the ProDataSet object from which to create the new ProDataSet object.

src-dataset-name

The name of the ProDataSet object from which to create the new ProDataSet object.

name-prefix

A character expression to prepend to each of the source ProDataSet member buffer names, which creates a new name for each new member buffer.

For a table handle, this method copies the field definitions from the specified source table and establishes the default or specified source indexes. You cannot call this method after another definitional method is called unless you call CLEAR( ) first.

For a ProDataSet object handle, this method creates a dynamic ProDataSet object like another static or dynamic ProDataSet object. Progress creates the new ProDataSet object with the same name, temp-table definitions, and relation definitions. Progress also creates the same before-image and after-image tables, if any exist for the source object. No data from the source temp-tables is copied. Progress also lets you rename the newly created ProDataSet member buffers by prepending a prefix to the source buffer names.

CREATE-NODE( ) method

Create an XML node in the current document. The first parameter must be a valid X-noderef handle and will refer to the new XML node if the method succeeds. This method merely creates the XML node as part of the XML document. The INSERT-BEFORE or APPEND-CHILD methods are required to actually insert it into the document's tree.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
CREATE-NODE( x-node-handle , name , type )
```

x-node-handle

A valid X-noderef handle to use for the new XML node.

name

A character expression representing the NAME of the node. The relationship between the node NAME and SUBTYPE attributes is shown in Table 61.

type

A character expression representing the node's SUBTYPE, which will be one of: ATTRIBUTE, CDATA-SECTION, COMMENT, DOCUMENT-FRAGMENT, ELEMENT, ENTITY-REFERENCE, PROCESSING-INSTRUCTION, TEXT.

Table 61: Relationship between the SUBTYPE and NAME attributes *(1 of 2)*

If the SUBTYPE is . . .	then the NAME attribute is . . .
ATTRIBUTE	The name of the attribute.
CDATA-SECTION COMMENT DOCUMENT-FRAGMENT TEXT	A constant value; the *name* parameter is ignored.
ELEMENT	The name of the XML tag, with any namespace prefix included.

Table 61: Relationship between the SUBTYPE and NAME attributes *(2 of 2)*

If the SUBTYPE is . . .	then the NAME attribute is . . .
ENTITY-REFERENCE	The name of the entity referenced without leading ampersand and trailing semicolon.
PROCESSING-INSTRUCTION	The target; the first token following the <? markup.

The following example demonstrates creating a node in a document. If hDoc is an X-document handle, and hNoderef and hNoderefChild are X-noderefs, this is how you would add hNoderefChild to hNoderef in the document associated with hDoc:

```
/* Assume hNoderef has previously been added to the tree */
/* Create a 4GL handle that can refer to a node in an XML parse tree.*/
CREATE X-NODEREF hNoderefChild.

/* Create an XML node whose name is "Address" & whose type is "ELEMENT"*/
hDoc:CREATE-NODE(hNoderefChild,"Address","ELEMENT")

/* Put this child into the tree and ultimately into the document. */
hNoderef:APPEND-CHILD(hNoderefChild).
```

CREATE-NODE-NAMESPACE( ) method

Creates a namespace-aware XML node whose name can be either a single string `y` or an `x:y` combination.

Note: To ensure consistency across all nodes in an XML document, use either the CREATE-NODE-NAMESPACE( ) method or the CREATE-NODE( ) method to build an XML document; do not use both methods within a single document.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
CREATE-NODE-NAMESPACE( x-node-handle , namespace-uri , qualified-name , type )
```

x-node-handle

A valid X-noderef handle to use for the new namespace-aware XML node.

namespace-uri

A character expression representing the namespace Uniform Resource Identifier (URI). The *namespace-uri* must be unique and persistent. Although the *namespace-uri* may be an HTTP URL, there is no guarantee that it points to a retrievable resource. It is only a name and care should be taken if you use this name for other purposes.

If the character expression evaluates to either the empty string ("") or the Unknown value (?), no namespace is associated with the element.

qualified-name

A character expression representing the name of the node, optionally qualified with a prefix including a colon (for example, *prefix:node-name*). Unless you are using a default namespace, a prefix is required and should be set to the prefix specified when you declared the namespace using the xmlns attribute.

type

A character expression representing the node's SUBTYPE, which will be either ELEMENT or ATTRIBUTE.

Example The following code fragment illustrates how to create a namespace-aware node in an XML document using either a specific namespace or the default namespace:

```
. . .
   /* Look for a colonized name in rootNodeName. */
   found = INDEX(rootNodeName, ":").
   IF found > 0 THEN
     DO:
       /* Namespace declarations are special kinds of attributes that */
       /* belong in the http://www.w3.org/2000/xmlns/ namespace.*/
       errStat = hDocument:CREATE-NODE-NAMESPACE(hNsDecl,
       "http://www.w3.org/2000/xmlns/", "xmlns:" +
       SUBSTRING(rootNodeName, 1, found - 1), "attribute").
     END.
   ELSE
     DO:
       /* Use the default namespace, which does not need a */
       /* namespace declaration prefix, and assign it to the */
       /* http://www.w3.org/2000/xmlns/ namespace.*/
       errStat = hDocument:CREATE-NODE-NAMESPACE(hNsDecl,
       "http://www.w3.org/2000/xmlns/", "xmlns", "attribute").
     END.
   IF errStat = NO THEN
      LEAVE.

   /* Set the value of the namespace attribute to the namespace URI. */
   hNsDecl:NODE-VALUE = namespaceURI.
. . .
```

CREATE-RESULT-LIST-ENTRY( ) method

Creates an entry in the result list for the current row. The developer uses the CREATE-RESULT-LIST-ENTRY method in conjunction with new browse rows or new query rows to synchronize the data with the query.

Return type: LOGICAL

Applies to: BROWSE widget, Query object handle

Syntax

```
CREATE-RESULT-LIST-ENTRY ( )
```

For browses, this method should be used with a ROW-LEAVE trigger when the NEW-ROW attribute is TRUE. For example, if the user adds a row to an updateable browse, you can use this method to create an entry for the new row in the result list.

CURRENT-CHANGED attribute

Indicates whether a record in a buffer is different following a FIND CURRENT or GET CURRENT statement or method. If the record is different, CURRENT-CHANGED is TRUE. Otherwise, CURRENT-CHANGED is FALSE.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle

Note: The CURRENT-CHANGED attribute corresponds to the CURRENT-CHANGED function.

CURRENT-COLUMN attribute

The value of the browse column that contains the current cell. This attribute moves focus to the cell in the specified column in the current row.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: BROWSE widget

For the browse, if the browse or a browse component currently has focus, then setting the attribute to another column causes the proper LEAVE and cell ENTRY events to happen.

If the setting of the CURRENT-COLUMN attribute happens when focus is outside of the browse, then the browse's internal handle to the current column is updated so that it will become the current column when you tab back into the browse. Also if you apply "START-SEARCH" the search mode will now use this column to search on.

CURRENT-ENVIRONMENT attribute

Returns a list of CGI environment variable settings and HTTP header information, and is used by the `get-cgi` WebSpeed API function. Intended for internal use only.

Data type: CHARACTER

Access: Readable

Applies to: WEB-CONTEXT system handle

CURRENT-ITERATION attribute

A widget handle for the current iteration of the frame or dialog box.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

This attribute is a read-only attribute for dialog boxes.

CURRENT-RESULT-ROW attribute

The sequence number of the current row of a dynamic query's result list.

Data type: INTEGER

Access: Readable

Applies to: Query object handle

Note: The CURRENT-RESULT-ROW attribute corresponds to the CURRENT-RESULT-ROW function.

See also CURRENT-RESULT-ROW function

CURRENT-ROW-MODIFIED attribute

Indicates whether any cells in the current row have been changed.

Data type: LOGICAL

Access: Readable

Applies to: BROWSE widget

The CURRENT-ROW-MODIFIED attribute is set to TRUE if the user has modified any cell within the current row since focus moved to that row.

CURRENT-WINDOW attribute

A current window for the specified procedure.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: THIS-PROCEDURE system handle (and all procedure handles)

Specifies and allows you to reset the current window used to parent alert box, dialog box, or frame widgets for the specified procedure. The default value is the Unknown value (?). Returns the Unknown value (?) for a Web service procedure or proxy persistent procedure.

If you set this attribute to the widget handle of a window, this value takes precedence over the CURRENT-WINDOW handle to provide the default window for parenting alert boxes, frames, and dialog boxes created within the procedure.

This attribute is especially useful for creating and associating a unique current window with each instantiation of a persistent procedure. For more information on persistent procedures, see the RUN statement reference entry.

CURSOR-CHAR attribute

The current character position of the text cursor on the current text line in an editor widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget

Assigning a value to CURSOR-CHAR moves the text cursor to the specified character position on the current text line. If the editor widget is not already realized, Progress realizes the widget when you query the CURSOR-CHAR attribute.

CURSOR-LINE attribute

The line within an editor widget where the text cursor is positioned.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget

Assigning a value to CURSOR-LINE moves the text cursor to the specified line. If the editor widget is not already realized, Progress realizes the widget when you query the CURSOR-LINE attribute.

CURSOR-OFFSET attribute

The character offset of the cursor within a widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Assigning a value to CURSOR-OFFSET moves the text cursor to the specified character offset. If the editor widget is not already realized, Progress realizes the widget when you query the CURSOR-OFFSET attribute.

In Windows, both the regular editor and the large editor support CURSOR-OFFSET.

For browse-columns, CURSOR-OFFSET specifies the character offset of the cursor within a browse-cell of the browse-column.

DATA-ENTRY-RETURN attribute

The behavior of the **RETURN** key for the fill-in widgets of a frame.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, the **RETURN** key in a fill-in acts like a **TAB**, and if the fill-in is the last widget in the tab order of its parent frame and of all ancestor frames, the **RETURN** key applies a GO event to the frame (behavior prior to Version 7). This GO event, from a fill-in **RETURN**, propagates to all ancestor frames and their descendants, including siblings of the current frame and their descendants, all in the same frame family. If a widget is not a fill-in, the window system handles **RETURN** entries.

The default value is TRUE for character interfaces and FALSE for graphical interfaces.

Progress ignores this attribute if there is a default button on the frame.

DATA-SOURCE attribute

Returns the handle to the Data-source object currently attached to the ProDataSet object buffer.

Data type: HANDLE

Access: Readable

Applies to: Buffer object handle

DATA-SOURCE-COMPLETE-MAP attribute

Returns a comma-separated list of field name pairs for all fields in a ProDataSet temp-table buffer that are mapped to corresponding fields in an attached Data-source object.

Data type: CHARACTER

Access: Readable

Applies to: Buffer object handle

This list is formatted as a comma-separated list of field name pairs, qualified with the ProDataSet and Data-source temp-table names, using the following syntax:

```
tt-buffer-name.tt-field-name,db-table-name.db-field-name [, ...]
```

Note: You may use a subscript reference for array fields mapped explicitly through subscripts.

If the ProDataSet temp-table buffer does not have an attached Data-source object, this attributes returns the Unknown value (?).

Use the ATTACHED-PAIRLIST attribute to get a list of only the field name pairs you specified with the most recently attached Data-source object.

DATA-SOURCE-MODIFIED attribute

Indicates that data in the data source associated with a ProDataSet temp-table buffer has been modified.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

- The DATA-SOURCE-MODIFIED attribute corresponds to the DATA-SOURCE-MODIFIED function.
- Progress sets this attribute from the SAVE-ROW-CHANGES( ) method. You can also set this attribute, if needed.
- This attribute is marshalled between the client and the AppServer.

DATA-TYPE attribute

A character value that represents the data type of the field associated with the widget. For example, the DATA-TYPE attribute of a slider widget always returns the value "INTEGER" because slider widgets can only represent integers.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (column), Buffer-field object handle, COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

This attribute is writeable for combo-boxes (only before realization), fill-ins, and text widgets only. For combo-boxes, writing to this attribute makes the drop-down list empty.

You must define this attribute as “CHARACTER” for SIMPLE and DROP-DOWN combo-boxes.

For widgets like image or rectangle, where a data type has no meaning, the attribute returns "UNKNOWN".

The DATA-TYPE attribute is only writable for dynamic fill-ins before they are realized. This attribute is read only for static fill-ins.

DATASET attribute

Returns the handle for the ProDataSet object of which the buffer is a member. Use this handle to access the attributes and methods of the associated ProDataSet object.

Data type: HANDLE

Access: Readable

Applies to: Buffer object handle

DATE-FORMAT attribute

The format used to represent dates during the current OpenEdge session (for the DATE, DATETIME, and DATETIME-TZ data types).

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

Typical values are "**mdy**" or "**dmy**". This attribute provides the same functionality as the Date Format (`-d`) parameter.

DBNAME attribute

The logical name of the database from which the field is taken.

Data type: CHARACTER

Access: Readable

Applies to: Buffer object handle, Buffer-field object handle, COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

The default logical name for a database name is the name of the database file without the file extension. You can set the logical name of a database using the Logical Database Name (-ld) parameter.

DB-REFERENCES attribute

A comma-separated list of the databases, (in the form of logical database names) referenced by a `.r` file or by a persistent procedure. Returns the Unknown value (`?`) for a Web service procedure or proxy persistent procedure.

Data type: CHARACTER

Access: Readable

Applies to: THIS-PROCEDURE system handle (and all procedure handles)

The following example displays a list of all databases referenced by sample.r in a comma-separated list that is contained in the DB-REFERENCES attribute.

```
RCODE-INFO:FILE-NAME = "sample.r".
DISPLAY RCODE-INFO:DB-REFERENCE.
```

DCOLOR attribute (Character Interfaces only)

The color number for the display color of the widget in character mode. This attribute is ignored in graphical interfaces.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For browse widgets, it is readable only. For a browse cell, it specifies the color of a specific cell in the view port. You can set this color only as the cell appears in the view port during a ROW-DISPLAY event.

For rectangles, DCOLOR specifies the fill color. In windows, DCOLOR specifies the color inherited by the menu bar, if the menu bar has no color specified.

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

You can now change the color of the background of menu frames (including menubars, submenus and pop-up menus) using the DCOLOR attribute. Previously, specifying the DCOLOR attribute for menus only changed the default color for menu items. Now, the DCOLOR attribute will be applied to the menu frame also. Note that no syntax changes were made. You can still specify the DCOLOR attribute for individual menu items.

For more information on widget color, see the PFCOLOR attribute.

DDE-ERROR attribute (Windows only)

The error condition returned by the most recent exchange in a DDE conversation associated with the frame. A DDE function or a DDE-NOTIFY event initiates an exchange in a DDE conversation.

Data type: INTEGER

Access: Readable

Applies to: FRAME widget

Table 62 lists the possible errors returned by an exchange in a DDE conversation.

Table 62: Progress DDE errors *(1 of 2)*

Error	Description
1	DDE INITIATE failure.
2	A DDE statement (DDE ADVISE, DDE EXECUTE, DDE GET, DDE REQUEST, or DDE SEND) time out.
3	Memory allocation error.
4	Invalid channel number (not an open conversation).
5	Invalid data item (in topic).
6	DDE ADVISE failure (data link not accepted).
7	DDE EXECUTE failure (commands not accepted).
8	DDE GET failure (data not available).
9	DDE SEND failure (data not accepted).

Table 62: Progress DDE errors *(2 of 2)*

Error	Description
10	DDE REQUEST failure (data not available).
11	DDE-NOTIFY event failure (data not available).
99	Internal error (unknown).

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DDE-ID attribute (Windows only)

The DDE channel number of the most recent conversation involved in an exchange.

Data type: INTEGER

Access: Readable

Applies to: FRAME widget

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DDE-ITEM attribute (Windows only)

The name of the data item affected by the most recent conversational exchange (for example, the name of a worksheet cell such as "R3C5").

Data type: CHARACTER

Access: Readable/Writeable

Applies to: FRAME widget

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DDE-NAME attribute (Windows only)

The name of the application involved in the most recent conversational exchange (for example, the name of a worksheet application such as "EXCEL").

Data type: CHARACTER

Access: Readable

Applies to: FRAME widget

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DDE-TOPIC attribute (Windows only)

The topic name of the most recent conversation (for example, the "System" topic, or the name of an Excel worksheet such as "Sheet1").

Data type: CHARACTER

Access: Readable

Applies to: FRAME widget

This attribute applies to any frame in Windows that is a Dynamic Data Exchange (DDE) frame for a DDE conversation.

DEBLANK attribute

How to process leading blanks in fill-in widgets during user input.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FILL-IN widget

This attribute applies to fill-ins for CHARACTER fields that are enabled for input.

If the DEBLANK attribute is TRUE, Progress removes leading blanks from the widget following user input that changes the SCREEN-VALUE attribute of the widget. Any leading blanks in the SCREEN-VALUE before input are not removed unless the user modifies the field. After the field is modified, the procedure must explicitly redisplay the field to view the effect of the DEBLANK attribute.

DEBUG( ) method

Starts and initializes the Debugger, and immediately gives control to the Debugger in stand-alone mode while blocking the invoking procedure.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
DEBUG ( )
```

This method has the same effect as starting OpenEdge with the Debugger (-debug) startup parameter, except that instead of running the Debugger from the OpenEdge command line, it runs it from the invoking procedure. The invoking procedure then waits to continue execution until the Debugger exits. Although the Debugger has no control over the invoking procedure, it can control any other procedure started with the Debugger RUN option.

Note: To use this method, you must have the Application Debugger installed in your OpenEdge environment.

If the Debugger starts successfully, this method returns TRUE after the Debugger exits. Otherwise, it returns FALSE with no effect.

For more information about running the Debugger in stand-alone mode, see *OpenEdge Development: Debugging and Troubleshooting*. For more information about the DEBUGGER system handle, see the reference entry for the DEBUGGER system handle.

DEBUG-ALERT attribute

Indicates whether Progress provides access to 4GL stack trace information when an error occurs during a session (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

When an error occurs in an interactive session, for any 4GL client, Progress displays an alert box with a help button that displays a dialog box containing the 4GL stack trace information.

You can use the DEBUG-ALERT attribute on the SESSION system handle with the `-clientlog` startup parameter to access 4GL stack trace information during a session.

If you set DEBUG-ALERT to TRUE and use the `-clientlog` startup parameter, the 4GL stack trace is written to the log file you specify with `-clientlog`. Messages are written to the specified log file as they are encountered during a session. This applies to both interactive and batch sessions. For WebSpeed and AppServer applications, the output goes to their respective log files.

Note: If you do not use the `-clientlog` startup parameter to specify a log file name, messages are not written to a log file.

You can also set DEBUG-ALERT to TRUE using the `-debugalert` startup parameter. For more information about the `-clientlog` and the `-debugalert` startup parameters, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

DECIMALS attribute

Indicates the number of decimal places, after the decimal point, that are stored for a buffer-field object that corresponds to a DECIMAL field. If the value of DECIMALS is nonzero, Progress rounds off any source that you assign to BUFFER-VALUE to the specified number of decimal places before completing the assignment. Otherwise, the assignment executes without rounding off the source value.

Note: Progress determines the number of decimal places to display, as opposed to store, from the FORMAT attribute (not the DECIMALS attribute) of the buffer-field.

Data type: INTEGER

Access: Readable

Applies to: Buffer-field object handle

DECLARE-NAMESPACE( ) method

Adds a namespace declaration to a tag in the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
DECLARE-NAMESPACE( namespace-URI [ , prefix ] )
```

namespace-URI

A LONGCHAR expression evaluating to the URI of the attribute.

prefix

A LONGCHAR expression evaluating to the prefix of the namespace.

Call this method to add a namespace declaration to a start tag. You can only call this method directly following a call to START-ELEMENT, WRITE-EMPTY-ELEMENT, INSERT-ATTRIBUTE, or DECLARE-NAMESPACE method. That is, you can only call this method when the WRITE-STATUS attribute is SAX-WRITE-TAG. The WRITE-STATUS attribute remains SAX-WRITE-TAG after this method call.

Regardless of the value of the STRICT attribute, this method fails if you do not call it after one of the valid methods listed above.

If you use an empty string (`""`) or the Unknown value (`?`) for the prefix, or you omit the prefix, then the method declares the default namespace: `xmlns="namespace-URI"`.

DEFAULT attribute

Indicates whether the button is a default button.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget

If the DEFAULT attribute is TRUE, the specified button is a default button. To make the specified button the default button for the frame, you must also set the frame's DEFAULT-BUTTON attribute to the widget handle of the button. A default button is one that handles all RETURN events when no other RETURN-enabling widget in the frame or dialog box has focus. RETURN-enabling widgets include any field-level widget for which a RETURN trigger is defined, or any button, whether or not it has a trigger defined. Thus, if a button has focus, that button handles the next RETURN event. If any other field-level widget without a RETURN trigger has focus, the default button handles the next RETURN event.

You can set this attribute only before the widget is realized.

Note: When the frame receives a default RETURN event, it actually sends a CHOOSE event to the default button. If the user presses the **RETURN** key while in a frame that has no default button, and the frame is part of a frame family, Progress applies the CHOOSE event to the first default button it can find within the frame family in random order.

DEFAULT-BUFFER-HANDLE attribute

Like static temp-tables, every dynamic temp-table is created with at least one buffer. This buffer's object handle is returned by this attribute. DEFAULT-BUFFER-HANDLE cannot be called until the TEMP-TABLE-PREPARE( ) method has been called, since the default buffer is not created until then.

Data type: HANDLE

Access: Readable

Applies to: Temp-table object handle

Syntax

```
tt-buffer-handle = tt-handle:DEFAULT-BUFFER-HANDLE
```

DEFAULT-BUTTON attribute

Indicates whether a button is a default button for the frame or dialog box.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

To make the specified button the default button for the frame or dialog box, you must also set the button's DEFAULT attribute to TRUE.

The DEFAULT-BUTTON attribute for frames is not supported in character mode.

The default button cannot display an image.

A default button is one that handles all RETURN events when no other RETURN-enabling widget in the frame or dialog box has focus. RETURN-enabling widgets include any field-level widget for which a RETURN trigger is defined, or any button, whether or not it has a trigger defined. Thus, if a button has focus, that button handles the next RETURN event. If any other field-level widget without a RETURN trigger has focus, the default button handles the next RETURN event.

Note: When the frame receives a default RETURN event, it actually sends a CHOOSE event to the default button. If the user presses the **RETURN** key while in a frame that has no default button, and the frame is part of a frame family, Progress applies the CHOOSE event to the first default button it can find within the frame family in random order.

DEFAULT-COMMIT attribute (AppServer only)

Indicates how an open transaction under the control of a transaction initiating procedure is to complete if the procedure is deleted in the absence of any SET-COMMIT( ) method or SET-ROLLBACK( ) method.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Transaction object handle

Setting the DEFAULT-COMMIT attribute to TRUE ensures that the transaction is completed. Setting it to FALSE ensures that the transaction is rolled back. The default value is FALSE.

Note: One common event that can terminate an open transaction is deleting the transaction initiating procedure that created the transaction.

DELETE( ) method

Deletes an item from a combo box, radio-set, or selection list.

Return type: LOGICAL

Applies to: COMBO-BOX widget, RADIO-SET widget, SELECTION-LIST widget

Combo-box and Selection-list Syntax

```
DELETE ( list-index | list-item )
```

list-index

An INTEGER expression that specifies the ordinal position of a value in the combo box list or selection list.

list-item

A character-string expression that specifies a single value or a delimiter-separated list of values in the widget.

The DELETE( ) method removes the item specified by *list-index*, or removes the specified *list-item* from the list. *list-item* can represent multiple items. For example, you might specify DELETE("Chicago, Boston, New York"), where the delimiter is a comma. The delimiter is a comma by default or is specified by the DELIMITER attribute. If the method is successful, it returns TRUE.

Radio-set syntax

```
DELETE ( label )
```

label

A character-string expression that specifies an item to delete from the radio-set.

The DELETE( ) method deletes the item from the radio-set, whose appearance changes depending on the user interface and the setting of the AUTO-RESIZE attribute. For all user interfaces, if AUTO-RESIZE is TRUE, the remaining items collapse toward the top to fill the gap left by the deleted item. If AUTO-RESIZE is FALSE in Windows, the remaining items are repositioned to evenly span the original radio-set dimensions; in character interfaces, the remaining items collapse upward as when AUTO-RESIZE is TRUE.

If the method is successful, it returns TRUE.

Note: A single call to DELETE can delete one or more items from a combo box or selection list, or one item from a radio set.

DELETE-CHAR( ) method

Deletes the character at the current text cursor position.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
DELETE-CHAR ( )
```

If the character is successfully deleted, the method returns TRUE.

DELETE-CURRENT-ROW( ) method

Deletes the most recently selected row from a browse and the results list.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DELETE-CURRENT-ROW ( )
```

This method does **not** delete the record from the database and has no effect on the database buffer. If you then want to delete the database record associated with the row, use the DELETE statement.

If the row is successfully deleted from the browse and results list, the method returns TRUE.

DELETE-HEADER-ENTRY( ) method

Deletes the XML underlying a SOAP-header-entryref object, without deleting the object.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
DELETE-HEADER-ENTRY ( )
```

DELETE-LINE( ) method

Deletes the line that currently contains the text cursor.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
DELETE-LINE ( )
```

If the line is successfully deleted, the method returns TRUE.

DELETE-NODE( ) method

Unlinks and deletes the node and its sub-tree from the XML document. The Progress handle is not deleted.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
DELETE-NODE( )
```

The following example demonstrates the use of the DELETE-NODE( ) method. Only use this when you are through using the node and all of its descendants.

```
hOldNode:DELETE-NODE( ).
```

DELETE-RESULT-LIST-ENTRY( ) method

Deletes the current row of a query's result list.

Return type: LOGICAL

Applies to: BROWSE widget, Query object handle

Syntax

```
DELETE-RESULT-LIST-ENTRY ( )
```

For the browse, DELETE-RESULT-LIST-ENTRY( ) solves the following problem: Suppose you create a browse with a primary table and a secondary table, and in the primary table, the key to the secondary table changes. Progress never displays the new secondary table because the result list entry contains the rowid of the original secondary table.

When you use DELETE-RESULT-LIST-ENTRY( ) together with CREATE-RESULT-LIST-ENTRY( ), you can update the result list entry and display the modified row without having to reopen the query.

For example, suppose you create a browse with customer.name, customer.salesrep, and salesrep.repname (from the Sports database). Then, in one record of the browse, you change customer.salesrep from “bbb” to “dkp.”

Without using DELETE-RESULT-LIST-ENTRY( ), the secondary record remains “bbb” until the query is reopened.

The following code fragment uses DELETE-RESULT-LIST-ENTRY( ) and CREATE-RESULT-LIST-ENTRY( ) to display the modified secondary record:

```
ON ROW-LEAVE OF my-brow DO:
    DEFINE VAR num AS INTEGER.
    DEFINE VAR ok AS LOGICAL.
    IF customer.sales-rep:MODIFIED = TRUE THEN DO:
        num = customer.cust-num.
        ok = my-brow:DELETE-RESULT-LIST-ENTRY().
        /* DELETE-RESULT-LIST-ENTRY() disconnects recs from rec bufs */
        /* so reread customer and salesrep records */
        FIND customer WHERE customer.cust-num EQ Num.
        FIND salesrep WHERE salesrep.sales-rep EQ
            customer.sales-rep:SCREEN-VALUE.
        /* Create new result list entry */
        /* with "new" secondary table's rowid */
        ok = my-brow:CREATE-RESULT-LIST-ENTRY().
        /* Update viewport */
        DISPLAY salesrep.rep-name WITH BROWSE my-brow.
    END.
END.
```

Note: During this operation, the query pointer must not move.

DELETE-SELECTED-ROW( ) method

Deletes the *n*th selected row from a browse and the results list.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DELETE-SELECTED-ROW ( n )
```

n

An integer expression that specifies a selected row within the browse.

Note: Do not confuse the DELETE-SELECTED-ROW method (note the singular) with the DELETE-SELECTED-ROWS method (note the plural).

Progress maintains a numbered list of selected rows, starting at 1. When the DELETE-SELECTED-ROW(*n*) method is encountered, Progress searches this list to find the *n*th selected row.

This method does **not** delete the record from the database and has no effect on the database buffer. If you want to delete the database record associated with the row, use the DELETE statement.

If the row is successfully deleted, the method returns TRUE.

If you want to delete all selected rows, whether it is one or many, DELETE-SELECTED-ROWS is the preferred, optimized method for doing so.

DELETE-SELECTED-ROWS( ) method

Deletes all currently selected rows from a browse and the associated results list.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DELETE-SELECTED-ROW ( )
```

Note: Do not confuse the DELETE-SELECTED-ROW method (note the singular) with the DELETE-SELECTED-ROWS method (note the plural).

This method does **not** delete the record from the database and has no effect on the database buffer. If you want to delete the database record associated with the row, use the DELETE statement.

If the row is successfully deleted, the method returns TRUE.

DELIMITER attribute

The character that separates values input to or output from a combo box or selection list.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: COMBO-BOX widget, RADIO-SET widget, SELECTION-LIST widget

Delimiter character can have any ASCII value from 1 to 127. The default delimiter is a comma.

DESELECT-FOCUSED-ROW( ) method

Deselects the row with current focus.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DESELECT-FOCUSED-ROW ( )
```

This method is ignored on single-select browse widgets, because focus follows selection.

DESELECT-ROWS( ) method

Deselects all currently selected rows in the browse and clears the associated record buffer.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DESELECT-ROWS ( )
```

If the rows are successfully deselected, the method returns TRUE.

DESELECT-SELECTED-ROW( ) method

Deselects the *n*th selected row in a browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
DESELECT-SELECTED-ROW ( n )
```

n

An integer expression that specifies a selected row within the browse.

Progress maintains a numbered list of selected rows, starting at 1. When the DESELECT-SELECTED-ROW(*n*) method is encountered, Progress searches this list to find the *n*th selected row. If the row is successfully deselected, the method returns TRUE.

DETACH-DATA-SOURCE( ) method

Detaches a Data-source object from a temp-table buffer in a ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
buffer-handle:DETACH-DATA-SOURCE( )
```

buffer-handle

A handle to the temp-table buffer in the ProDataSet object.

See also ATTACH-DATA-SOURCE( ) method

DISABLE( ) method

Disables the radio set button.

Return type: LOGICAL

Applies to: RADIO-SET widget

Syntax

```
DISABLE ( label )
```

label

A character-string expression that specifies the label of a button in the radio set.

If the operation is successful, the method returns TRUE.

DISABLE-AUTO-ZAP attribute

Indicates whether Progress ignores the value of the AUTO-ZAP attribute.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget (column), COMBO-BOX widget, FILL-IN widget

If DISABLE-AUTO-ZAP is TRUE, Progress ignores the value of the AUTO-ZAP attribute and assumes it is FALSE. If the DISABLE-AUTO-ZAP attribute is FALSE, Progress assumes the value of the AUTO-ZAP attribute is TRUE.

DISABLE-CONNECTIONS( ) method

Indicates that Progress no longer listen for or accept new connections on the port associated with the server socket. However, all existing connections are still valid.

Return type: LOGICAL

Applies to: Server socket object handle

Syntax

```
DISABLE-CONNECTIONS( )
```

Returns TRUE if connections are disabled for this server socket object, even if the server socket object was never enabled or was already disabled.

DISABLE-DUMP-TRIGGERS( ) method

Allows a user to access a buffer object's table without firing FIND triggers.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
DISABLE-DUMP-TRIGGERS ( )
```

To run this method, the end user must have CAN-DUMP privileges for the table.

Triggers cannot be disabled from a persistent procedure.

The FIND trigger remains disabled until the procedure in which it is disabled returns.

See also DISABLE TRIGGERS statement

DISABLE-LOAD-TRIGGERS( ) method

Allows a user to subsequently create or update a buffer object's table without firing update triggers such as CREATE, WRITE, ASSIGN, or DELETE.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
DISABLE-LOAD-TRIGGERS (allow-replication)
```

allow-replication

A logical expression that allows the following replication triggers to run when you set *allow-replication* to true: REPLICATION-CREATE, REPLICATION-DELETE, and REPLICATION-WRITE.

To run this method, the end user must have CAN-LOAD privileges for the table.

Triggers cannot be loaded from a persistent procedure.

The triggers remain disabled until the procedure in which it is disabled returns.

See also DISABLE TRIGGERS statement

DISCONNECT( ) method

Disconnects the client from the AppServer or Web service currently associated with the specified server handle. For the socket, closes the socket by terminating the connection between the socket and the port to which it is connected.

Return type: LOGICAL

Applies to: Server object handle, Socket object handle

Syntax

```
DISCONNECT ( )
```

When the AppServer receives a disconnect request:

- **For a state-reset or state-aware AppServer** — Control returns immediately to the client application, and any configured Disconnect procedure executes in the connected AppServer agent.
- **For a stateless AppServer** — Any configured Disconnect procedure executes in an available AppServer agent, then control returns to the client application.
- **For any AppServer with pending asynchronous requests** — All running or pending asynchronous requests are cancelled and the corresponding event procedure is called for each request. The CANCELLED attribute on the asynchronous request handle for all such cancelled requests is set to TRUE.

If an error occurs during the disconnection from an AppServer, DISCONNECT( ) returns FALSE. An error occurs if:

- The server handle is invalid.
- The server is either not connected or already disconnected.

If DISCONNECT( ) completes successfully, the CONNECTED( ) method returns FALSE. In addition, all attributes of the SERVER object (except for the FIRST PROCEDURE and LAST PROCEDURE attributes) are restored to their initial state. Specifically, SUBTYPE returns the Unknown value (?).

For more information on AppServers, see *OpenEdge Application Server: Developing AppServer Applications*.

For sockets, Progress automatically closes a socket if it detects that the connection to which the socket is bound has failed or been terminated.

DISPLAY-MESSAGE( ) method

Displays a message in an alert box. The Debugger stores these messages and displays them to the user in an alert box when the Debugger regains control of an application.

Return type: INTEGER

Applies to: DEBUGGER system handle

Syntax

```
DISPLAY-MESSAGE ( char-expression )
```

char-expression

Any character expression.

This method appends a new line to *char-expression* before displaying the specified string. If the Debugger is initialized and *char-expression* is a valid character expression, this method returns TRUE. Otherwise, it returns FALSE with no effect.

Note: To use this method, you must have the Application Debugger installed in your OpenEdge environment.

DISPLAY-TIMEZONE attribute

The time zone offset, in minutes, used to display DATETIME-TZ data. The default value is the session's time zone offset.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

If the format string for the DATETIME-TZ does not include the time zone offset, Progress converts the DATETIME-TZ data to the local date and time and displays the data with the time zone offset specified by this attribute.

If the format string for the DATETIME-TZ includes the time zone offset, Progress ignores this attribute and displays the data in the local time of the value, along with the time zone offset.

Set this attribute to the Unknown value (?) to use the session's time zone offset for display.

DISPLAY-TYPE attribute

The type of display used in the session—"GUI" for a graphical display and "TTY" for a character-mode display.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

DOMAIN-DESCRIPTION attribute

The description of the authentication domain that authenticated the user represented by the Client-principal object. If not specified, Progress returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the Client-principal object is sealed, this attribute is read-only.

See also DOMAIN-NAME attribute, DOMAIN-TYPE attribute

DOMAIN-NAME attribute

The name of the authentication domain that authenticated the user represented by the Client-principal object. You must set this attribute before you can seal the associated Client-principal object using the SEAL( ) method.

The authentication domain name you specify must match an authentication domain name registered in the trusted authentication domain registry for the Client-principal object. Progress uses this domain name to find the associated authentication domain registry entry to validate the object before you can use it.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

If you specify the Unknown value (?) or the empty string (""), Progress generates a run-time error.

Once the Client-principal object is sealed, this attribute is read-only.

See also DOMAIN-DESCRIPTION attribute, DOMAIN-TYPE attribute

DOMAIN-TYPE attribute

The type of the authentication domain that authenticated the user represented by the Client-principal object. If not specified, Progress returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the Client-principal object is sealed, this attribute is read-only.

See also DOMAIN-DESCRIPTION attribute, DOMAIN-NAME attribute

DOWN attribute (Graphical interfaces only)

Indicates the number of iterations in a down frame that contain data or number of potential rows in a browse widget. For a one-down frame, the value of DOWN is 1. Sets the number of browse-rows that appear in the viewport.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, FRAME widget

If you change the value of a browse's DOWN attribute, you change the number of rows that appear in the viewport, which might change the value of the browse's HEIGHT-CHARS and HEIGHT-PIXELS attributes. Changing the value of a browse's DOWN attribute does not change the value of the browse's ROW-HEIGHT-CHARS and ROW-HEIGHT-PIXELS attributes.

Note: If the browse's height is set with the DOWN attribute and a browse column is added using the ADD-CALC-COLUMN( ), ADD-COLUMNS-FROM( ) or ADD-LIKE-COLUMN( ) methods, the browse's height may change to ensure that the number of DOWN is preserved.

DRAG-ENABLED attribute

Indicates whether the user can simultaneously hold down the mouse select button and drag the mouse cursor through the selection list. As the mouse cursor passes over an item, the item is highlighted. When the user releases the select button, the highlighted item becomes the selected item.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SELECTION-LIST widget

In this style of selecting, a user can deselect an item only by selecting another. Once an item is selected, the list cannot revert to its unselected state.

The default value for this attribute is TRUE.

Note: In Windows, DRAG-ENABLED is always TRUE.

You can set this attribute only before the widget is realized.

DROP-TARGET attribute (Windows only; Graphical interfaces only)

Indicates whether the widget can accept dropped files.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

If DROP-TARGET is TRUE, dragging one or more files over the widget causes the mouse pointer to change to indicate that the widget can accept the files. The default value of DROP-TARGET is FALSE.

For related information, see the reference entries for the DROP-TARGET option of the DEFINE BROWSE statement, DEFINE BUTTON statement, DEFINE FRAME statement, and DEFINE VARIABLE statement.

DUMP-LOGGING-NOW( ) method

Writes the accumulated query statistics for the specified query to the log file.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
query-handle:DUMP-LOGGING-NOW(reset-expression)
```

query-handle

A variable of type HANDLE that represents the handle to a query object.

reset-expression

A logical expression where TRUE indicates that Progress clear the query statistics after writing the statistics to the log file, and FALSE indicates that Progress leave the query statistics unchanged.

If Progress writes the query statistics to the log file successfully, the method returns TRUE. If Progress does not write the query statistics to the log file, the method returns FALSE. Progress does not write query statistics to the log file under the following conditions:

- QryInfo logging for the specified query was not turned on before the query started.

  To turn on basic logging for an individual query, you must set the BASIC-LOGGING attribute to TRUE before a query starts. For a dynamically opened query, this is before the QUERY-PREPARE( ) method. For a statically opened query, this is before the OPEN QUERY statement.

- QryInfo logging was turned off for the specified query, or all queries, before the query completed.

You can use the DUMP-LOGGING-NOW( ) method only when the logging level of the QryInfo log entry type is set to level 2 (Basic) or higher. To set the logging level of the QryInfo log entry type, use the LOG-ENTRY-TYPES attribute or the Log Entry Types (`-logentrytypes`) startup parameter with the logging level option.

If the logging level of the QryInfo log entry type is set to level 2 (Basic), the DUMP-LOGGING-NOW( ) method writes query statistics to the log file only if the BASIC-LOGGING attribute is set to TRUE before the query started.

For more information about the Log Entry Types (`-logentrytypes`) startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

For more information about logging query statistics, see *OpenEdge Development: Debugging and Troubleshooting*.

DYNAMIC attribute

Indicates whether the widget is dynamic or static.

Data type: LOGICAL

Access: Readable

Applies to: BROWSE widget, Buffer object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, ProDataSet object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, Query object handle, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, Temp-table object handle, TEXT widget, TOGGLE-BOX widget, WINDOW widget

If the DYNAMIC attribute is TRUE, the widget is dynamic, otherwise it is static.

EDGE-CHARS attribute

The width, in character units, of the edge of a rectangle.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: RECTANGLE widget

EDGE-PIXELS attribute

The width, in pixels, of the edge of a rectangle.

Data type: INTEGER

Access: Readable/Writeable

Applies to: RECTANGLE widget

EDIT-CAN-PASTE attribute (Windows only; Graphical interfaces only)

Indicates whether the Clipboard contains data that can be pasted into the widget.

Data type: LOGICAL

Access: Readable

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

If the Clipboard contains data that can be pasted into the widget, EDIT-CAN-PASTE is TRUE. Otherwise, it is FALSE.

EDIT-CAN-UNDO attribute (Windows only; Graphical interfaces only)

Indicates whether the widget can undo the last modification.

Data type: LOGICAL

Access: Readable

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget

If the widget can undo the last modification, EDIT-CAN-UNDO has the value TRUE, otherwise it has the value FALSE.

If you set EDIT-CAN-UNDO to any value, Progress empties the undo buffer.

EDIT-CLEAR( ) method (Windows only; Graphical interfaces only)

Deletes the selected text.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
EDIT-CLEAR ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

EDIT-COPY( ) method (Windows only; Graphical interfaces only)

Copies the currently selected text in the widget to the Clipboard.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
EDIT-COPY ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

EDIT-CUT( ) method (Windows only; Graphical interfaces only)

Copies the currently selected text in the widget to the Clipboard and then deletes the selected text.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
EDIT-CUT ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

EDIT-PASTE( ) method (Windows only; Graphical interfaces only)

Copies the currently selected text of the Clipboard into the widget at the current cursor position, if the Clipboard contains text data.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
EDIT-PASTE ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

EDIT-UNDO( ) method (Windows only; Graphical interfaces only)

Makes the editor undo its most recent edit if possible.

Return type: LOGICAL

Applies to: COMBO-BOX widget, EDITOR widget

Syntax

```
EDIT-UNDO ( )
```

If the widget performs the operation successfully, the method returns TRUE. Otherwise, it returns FALSE.

EMPTY attribute

Indicates whether the SCREEN-VALUE attribute for the editor contains text.

Data type: LOGICAL

Access: Readable

Applies to: EDITOR widget

The EMPTY attribute is TRUE if the editor contains no text (that is, the editor's SCREEN-VALUE is null).

EMPTY-DATASET( ) method

Empties a ProDataSet object of all records in its associated temp-tables.

Return type: LOGICAL

Applies to: ProDataSet object handle

Syntax

```
EMPTY-DATASET ( )
```

EMPTY-TEMP-TABLE( ) method

Deletes all records from a temporary table associated with a buffer object.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
EMPTY-TEMP-TABLE ( )
```

Note: This method corresponds to the EMPTY TEMP-TABLE statement.

ENABLE( ) method

Enables the specified radio button within the radio set.

Return type: LOGICAL

Applies to: RADIO-SET widget

Syntax

```
ENABLE ( label )
```

label

A character-string expression that specifies the label of a radio button.

If the operation is successful, the method returns TRUE.

ENABLE-CONNECTIONS( ) method

Specifies the TCP/IP port that Progress uses to listen for new connections. Once called, Progress automatically listens for and accepts new connections for the specified port.

Return type: LOGICAL

Applies to: Server socket object handle

Syntax

```
ENABLE-CONNECTIONS( connection-parms )
```

connection-parms

A character string expression that contains a space-separated list of one or more socket connection parameters.

Table 63 describes the socket connection parameters you can include in this string.

Table 63: Socket connection parameters *(1 of 2)*

Parameter	Description
`-S socket-port`	The TCP/IP port number that Progress should listen to and accept connections on. You can specify either an explicit port number or a TCP service name. If you use a TCP service name, the method uses the port number associated with that name in the TCP/IP services file.
`-pf filename`	Optional. A text file containing any of the socket connection parameters described in this table. If this file contains any other OpenEdge startup parameters, this method ignores them.

Table 63: Socket connection parameters *(2 of 2)*

Parameter	Description
`-qsize` *`backlog`*	Optional. The length of the pending-connection queue — that is, the maximum number of connection requests you want the server to queue while it processes the connections already accepted. If the queue is full when a connection request is received, it is refused. The default length of the queue depends on the platform. **Note:** On some platforms, the value you supply for *`backlog`* is modified by addition, subtraction, multiplication, division, or some combination of these, and it is this modified value that becomes the maximum length of the queue. For more information, see the documentation for your platform.
`-ssl`	If specified, the connection is SSL-based. **Note:** Be sure you need SSL before using this option. SSL incurs more or less heavy performance penalties, depending on resources and load.
`-keyalias` *`aliasname`*	Sets the alias name of the Public[/Private] key and digital certificate to use within the keystore. If not specified, the default `default_server` key alias is used.
`-keyaliaspasswd` *`encpwd`*	Sets the password to use in accessing the Public[/Private] key and digital certificate. Always specify a `-keyaliaspassword` when the `-keyalias` option is used. The default password only exists when using the `default_server` server certificate.
`-nosessioncache`	If specified, caching for the SSL client session is disabled.
`-sessiontimeout` [*`seconds`*]	The maximum amount of time, in seconds, that the server waits before it rejects a SSL client's request to resume a session. The default value is 180 seconds.

Neither an AppServer nor a WebSpeed agent can act as a socket server, since they are already listening on a port. ENABLE-CONNECTIONS is only valid from batch clients, GUI clients and character clients. This method will generate an error if it is called from an invalid application. This method will also generate an error if it is called multiple times without the DISABLE-CONNECTION( ) method being called in between.

Note: Connections to an SSL-enabled server socket require the management of public keys on the client (SSL client) and private keys on the server (SSL server). For 4GL sockets, the SSL client is the 4GL session initiating the SSL connection on a socket object and the SSL server is the 4GL session enabling SSL connections on a server socket object. For information on using SSL to secure a 4GL socket connection, see the sections on sockets in *OpenEdge Development: Programming Interfaces*. For more information on SSL and managing private key and digital certificate stores for OpenEdge SSL clients and servers, see *OpenEdge Getting Started: Core Business Services*.

ENABLE-EVENTS( ) method (Windows only)

Enables event notification for automation objects.

Return type: CHARACTER

Applies to: Automation Object

Syntax

```
ENABLE-EVENTS ( event-proc-prefix )
```

event-proc-prefix

A character-string expression that Progress prepends to event names. The resulting string is the name of the internal procedure Progress runs when an event is fired. During an event notification, all running procedures and all persistent procedures are searched to find a procedure with the name matching *event-proc-prefix.eventname* (for example, `ExcelWB.SelectionChanged`).

ENCODING attribute

Returns the name of the character encoding used to encode the contents of an XML document. The default encoding is UTF-8.

Data type: CHARACTER

Access: Readable/Writable

Applies to: X-document object handle, SAX-writer object handle

For an X-document object, Progress sets the ENCODING attribute to the encoding name specified in the XML document's encoding declaration when you load an XML document using the LOAD( ) method.

You can also set the ENCODING attribute to the name of the character encoding to use when saving an XML document using the SAVE( ) method. Progress records this character encoding in the encoding declaration in the XML document's prologue. If you do not set the ENCODING attribute, when you save the document, the document will not have an encoding declaration in its prologue, but the document will be saved with the default encoding of UTF-8.

For a SAX-writer object, you can set the ENCODING attribute to the name of the character encoding to use when writing the XML document. You can set this attribute only when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, you can only change this attribute when the writer is not writing, otherwise it fails and generates an error message. Progress records this character encoding in the encoding declaration in the XML document's prologue. If you do not set the ENCODING attribute, when you write the document, the document will not have an encoding declaration in its prologue, but the document will be written with the default encoding of UTF-8.

The encoding name must be an Internet Assigned Numbers Authority (IANA) name supported by the Progress XML Parser. Table 64 lists the names of the supported IANA encodings and their corresponding Progress code pages.

Table 64: IANA encodings and corresponding Progress code pages *(1 of 4)*

IANA encoding name	Progress code page name
Big5	BIG-5
EUC-JP	EUCJIS
GB_2312-80	GB2312

Table 64: IANA encodings and corresponding Progress code pages *(2 of 4)*

IANA encoding name	Progress code page name
GB18030	GB18030
GBK	CP936
hp-roman8	ROMAN-8
IBM00858	IBM858
IBM037	IBM037
IBM273	IBM273
IBM277	IBM277
IBM278	IBM278
IBM284	IBM284
IBM297	IBM297
IBM437	IBM437
IBM500	IBM500
IBM850	IBM850
IBM851	IBM851
IBM852	IBM852
IBM857	IBM857
IBM861	IBM861
IBM862	IBM862
IBM866	IBM866
ISO-8859-1	ISO8859-1
ISO-8859-2	ISO8859-2
ISO-8859-3	ISO8859-3

Table 64: IANA encodings and corresponding Progress code pages *(3 of 4)*

IANA encoding name	Progress code page name
ISO-8859-4	ISO8859-4
ISO-8859-5	ISO8859-5
ISO-8859-6	ISO8859-6
ISO-8859-7	ISO8859-7
ISO-8859-8	ISO8859-8
ISO-8859-9	ISO8859-9
ISO-8859-10	ISO8859-10
ISO-8859-15	ISO8859-15
KOI8-R	KOI8-R
KS_C_5601-1987	KSC5601
Shift_JIS	SHIFT-JIS
TIS-620	620-2533
US-ASCII	-cpinternal
UTF-16	UTF-16
UTF-32	UTF-32
UTF-8	UTF-8
windows-1250	1250
windows-1251	1251
windows-1252	1252
windows-1253	1253
windows-1254	1254
windows-1255	1255

Table 64: IANA encodings and corresponding Progress code pages *(4 of 4)*

IANA encoding name	Progress code page name
windows-1256	1256
windows-1257	1257
windows-1258	1258

ENCRYPT-AUDIT-MAC-KEY( ) method

Encrypts and encodes the specified character expression and returns an encrypted character value that you can store for later use in message authentication code (MAC) operations.

Return type: CHARACTER

Applies to: AUDIT-POLICY system handle

Syntax

```
ENCRYPT-AUDIT-MAC-KEY( encrypt-key )
```

encrypt-key

A character expression containing the key to encrypt. Progress converts this key to UTF-8 before encrypting it and storing it, which ensures a consistent value regardless of code page settings.

Example The following code fragment illustrates how to use the ENCRYPT-AUDIT-MAC-KEY( ) method:

```
DEF VAR val as CHAR.
DEF VAR key as CHAR init “Open Sesame”.
.
.
.
val = AUDIT-POLICY:ENCRYPT-AUDIT-MAC-KEY(key).
.
.
.
_db-detail._db-mac-key = val.
```

ENCRYPTION-SALT attribute

The default salt value (a random series of bytes) to use with the GENERATE-PBE-KEY function. The default value is the Unknown value (?), which indicates that no salt value is used to generate the password-based encryption key.

Data type: RAW

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

If specified, this salt value is combined with a password value and hashed some number of times to generate a password-based encryption key (using the algorithm specified by the PBE-HASH-ALGORITHM attribute and the number of iterations specified by the PBE-KEY-ROUNDS attribute).

When set, only the first 8 bytes are used. If the value has fewer than 8 bytes, it is padded at the end with zero-value bytes.

You can use the GENERATE-PBE-SALT function to generate a salt value, which can help to ensure that the password key value is unique.

You are responsible for generating, storing, and transporting this value.

See also GENERATE-PBE-KEY function, GENERATE-PBE-SALT function

END-DOCUMENT( ) method

Closes the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
END-DOCUMENT( )
```

Closes the XML stream. This is the logical conclusion of creating the XML document.

If you call this method before the START-DOCUMENT method, the method fails. WRITE-STATUS must not be SAX-WRITE-IDLE or SAX-WRITE-COMPLETE when you call this method. END-DOCUMENT changes WRITE-STATUS to SAX-WRITE-COMPLETE.

If the STRICT attribute is TRUE and the final tag has not been closed (that is, the root node), then this method fails.

See also START-DOCUMENT( ) method

END-ELEMENT( ) method

Ends an XML node based upon the name of the node with in a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
END-ELEMENT( name [, namespace-URI ] )
```

name

A LONGCHAR expression evaluating to the fully qualified or unqualified name of the element.

namespace-URI

A CHARACTER expression evaluating to the URI of the element. If the element doesn't contain a namespace, it can evaluate to an empty string ("") or the Unknown value (?).

Ends an XML node and sets the WRITE-STATUS to SAX-WRITE-ELEMENT.

For every invocation of END-ELEMENT, there must be a preceding corresponding call of the START-ELEMENT method. All the parameter values must match for the methods to correspond. The method does not resolve namespaces. Instead, it matches the namespace against the corresponding START-ELEMENT value.

If the STRICT attribute is TRUE and the method does not match a preceding START-ELEMENT call, then the method fails.

See also START-ELEMENT( ) method

END-EVENT-GROUP( ) method

Indicates the end of a group of related events in the current session.

Return type: LOGICAL

Applies to: AUDIT-CONTROL system handle

Syntax

```
END-EVENT-GROUP( )
```

After calling this method, the EVENT-GROUP-ID attribute is cleared for all connected audit-enabled databases and is no longer recorded in audit event records for this event group.

There can be only one active event group per session at any one point in time. To set a different event group for the session, you can:

- Call the END-EVENT-GROUP( ) method, to end the current event group, and then call the BEGIN-EVENT-GROUP( ) method to begin the new event group.
- Call the BEGIN-EVENT-GROUP( ) method to begin the new event group. If there is an existing event group in effect, Progress ends the existing event group before beginning the new event group.

Calling this method does not generate an audit event or an audit record.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

See also BEGIN-EVENT-GROUP( ) method, EVENT-GROUP-ID attribute

END-FILE-DROP( ) method (Windows only; Graphical interfaces only)

Terminates a drag-and-drop operation and frees the memory allocated by Windows to hold the names of the dropped files.

Return type: BOOLEAN

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
END-FILE-DROP ( )
```

This method returns TRUE if it is successful. If there is no current drag-and-drop operation, GET-DROPPED-FILE( ) returns FALSE.

END-USER-PROMPT attribute

A freeform string that WebClient uses when prompting for a userid and password, if it does not find those values in the security cache.

Data type: CHARACTER

Access: Readable

Applies to: CODEBASE-LOCATOR system handle

ENTRY( ) method

Returns the character-string value of the specified list entry.

Return type: CHARACTER

Applies to: COMBO-BOX widget, SELECTION-LIST widget

Syntax

```
ENTRY ( list-index )
```

list-index

An integer expression that specifies an entry within the combo-box list or selection list.

ENTRY-TYPES-LIST attribute

Returns a character string containing a comma-separated list of all valid entry types for the current OpenEdge environment.

Data type: CHARACTER

Access: Readable

Applies to: LOG-MANAGER system handle

ERROR attribute

A compile-time or run-time error condition.

Data type: LOGICAL

Access: Readable

Applies to: Asynchronous request object handle, Buffer object handle, COMPILER system handle, ERROR-STATUS system handle, ProDataSet object handle, Temp-table object handle

For the asynchronous request object handle, the ERROR attribute indicates that the ERROR condition was encountered during the processing of an AppServer or Web Server request. If the COMPLETE attribute is FALSE, the value of this attribute is the Unknown value (?). This attribute is set immediately before the event procedure is executed.

For the COMPILER system handle, the ERROR attribute indicates whether an error occurred in the preceding compilation. If no error occurred in the preceding compilation, the value of ERROR is the Unknown value (?).

For the ProDataSet object handle, Temp-table object handle, and Buffer object handle, the ERROR attribute indicates whether an error occurred during a FILL or SAVE-ROW-CHANGES operation. The ERROR attribute is writeable for a ProDataSet object.

The ERROR attribute corresponds to the ERROR function.

For a ProDataSet object handle or a Temp-table object handle, the ERROR attribute is reset to FALSE when one of the following methods removes the object's before-image records:

- FILL( ) method
- EMPTY-DATASET( ) method
- MERGE-CHANGES( ) method
- ACCEPT-CHANGES( ) method
- REJECT-CHANGES( ) method

For the Buffer object handle, this attribute causes the row to be backed out rather than merged with the MERGE-CHANGES( ) method or the MERGE-ROW-CHANGES( ) method.

This attribute is marshalled between the client and the AppServer.

For the ERROR-STATUS system handle, the ERROR attribute indicates whether the Progress ERROR condition was raised in the most recent statement that used the NO-ERROR option. If no ERROR condition was raised in that statement, the value of the ERROR attribute is FALSE.

Note: Statements such as COMPILE handle errors as part of their normal function that do not raise the Progress ERROR condition. For example, the COMPILE statement does not raise the ERROR condition when it encounters compilation errors in a procedure.

ERROR-COLUMN attribute

The character position at which a compiler error occurred.

Data type: INTEGER

Access: Readable

Applies to: COMPILER system handle

If no error occurred in the preceding compilation, the value of ERROR-COLUMN is the Unknown value (?).

ERROR-OBJECT-DETAIL attribute

Identifies the SOAP-fault object that contains SOAP fault message detail.

Data type: HANDLE

Access: Readable

Applies to: ERROR-STATUS system handle

If a Web service operation generates a SOAP fault message, Progress generates the following error:

```
Web service %s<operation> failed.  SOAP faultstring is %s (nnnn)
```

The complete SOAP fault error message is returned to the 4GL as part of the ERROR-STATUS system handle.

If the 4GL application invokes the Web service operation with the NO-ERROR option on the RUN statement, any errors that occur as a result of the operation are suppressed. In this case, the application can access the SOAP fault message detail using the SOAP-fault and SOAP–FAULT-DETAIL object handles. Otherwise, Progress displays the error message to the end user.

ERROR-ROW attribute

The line number at which a compiler error occurred.

Data type: INTEGER

Access: Readable

Applies to: COMPILER system handle

If no error occurred in the preceding compilation, the value of ERROR-ROW is the Unknown value (?).

ERROR-STRING attribute

An arbitrary string value associated with a buffer or temp-table object that provides descriptive information about an error on that object.

Data type: CHARACTER

Access: Readable/Writable

Applies to: Buffer object handle, Temp-table object handle

Setting the attribute does not in any way signal an error condition to Progress. Progress does not inspect this string, or take action based on its value.

Progress automatically clears this attribute by setting its value to the empty string (“”) when one of the following occurs:

- The FILL( ) method is used on any ProDataSet object containing a temp-table, on one of its member TEMP-TABLE objects, or on a parent buffer that cascades down through that TEMP-TABLE object.
- The EMPTY-TEMP-TABLE( ) method is used on a TEMP-TABLE object buffer.
- You set the ERROR-STRING attribute, for the TEMP-TABLE object, to the empty string ("").

The number of characters in this string is limited to 3K.

EVENT-GROUP-ID attribute

Returns the universally unique identifier (UUID) for the audit event group in effect for the current session, as a Base64 character string. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Data type: CHARACTER

Access: Readable

Applies to: AUDIT-CONTROL system handle

The value of this attribute is set by the BEGIN-EVENT-GROUP( ) method, and cleared by the END-EVENT-GROUP( ) method. This value is recorded in all audit event records generated for this audit event group until you either end the current audit event group or begin a different audit event group.

If no audit event group is in effect, this method returns the Unknown value (?).

See also BEGIN-EVENT-GROUP( ) method, END-EVENT-GROUP( ) method

EVENT-PROCEDURE attribute

The name of the internal procedure to run as the event procedure for an asynchronous request.

Data type: CHARACTER

Access: Readable

Applies to: Asynchronous request object handle

The name of this internal procedure is the same as the name of the event procedure as specified by the EVENT-PROCEDURE option on the RUN statement. If the EVENT-PROCEDURE option is not specified, this attribute is set to the empty string ("").

EVENT-PROCEDURE attribute

The name of an internal procedure you want Progress to execute when a dynamic, asynchronous invoke returns.

Note: Applies only if the ASYNCHRONOUS attribute is TRUE.

Return type: CHARACTER

Applies to: CALL object handle

Syntax

```
EVENT-PROCEDURE [ = event-internal-procedure ]
```

event-internal-procedure

A CHARACTER expression indicating the name of the event handler for a dynamic invoke that is asynchronous.

The default is the Unknown value (?).

Note: If the ASYNCHRONOUS attribute is TRUE, you must set EVENT-PROCEDURE before you execute the INVOKE( ) method.

EVENT-PROCEDURE-CONTEXT attribute

The procedure handle of the active procedure context where the event procedure is defined for an asynchronous request.

Data type: HANDLE

Access: Readable

Applies to: Asynchronous request object handle

This procedure handle is the same as the handle specified by the EVENT-PROCEDURE...IN *procedure-context* option of the RUN statement that executes this request. If the EVENT-PROCEDURE...IN option is not specified (the default), this attribute is set to the value of the THIS-PROCEDURE system handle at the time the RUN statement was executed.

EVENT-PROCEDURE-CONTEXT attribute

A handle to a running persistent procedure containing an internal procedure you want Progress to execute when a dynamic, asynchronous invoke returns.

Note: Applies only if the ASYNCHRONOUS attribute is TRUE.

Data type: HANDLE

Access: Readable/Writable

Applies to: CALL object handle

Syntax

```
EVENT-PROCEDURE-CONTEXT [ = handle-expression ]
```

handle-expression

A HANDLE expression indicating the handle of a running persistent procedure in whose context an event procedure is to run.

The default value is the Unknown value (?).

EVENT-TYPE attribute

The type of the last event.

Data type: CHARACTER

Access: Readable

Applies to: LAST-EVENT system handle

Valid event types are:

- **"KEYPRESS"** — When the detected event is a keyboard event identified by key label, such as **ESC**, **CTRL+A**, or **A**.
- **"MOUSE"** — When the detected event is a portable or three-button mouse event, such as MOUSE-SELECT-UP or LEFT-MOUSE-UP.
- **"PROGRESS"** — When the detected event is a high-level Progress event. These include all events identified as direct manipulation, key function, developer, and other miscellaneous events, such as SELECTION, DELETE-LINE, U1, or CHOOSE.

EXCLUSIVE-ID attribute

The ID assigned to a state-aware cookie. Intended for internal use only.

Data type: CHARACTER

Access: Readable

Applies to: WEB-CONTEXT system handle

EXPAND attribute

How to set the size of a horizontal radio set.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: RADIO-SET widget

This attribute applies to radio sets whose HORIZONTAL attribute are set to TRUE (for horizontal alignment).

If TRUE, the size for each button is equal to the width of the button with the longest label. If FALSE, the size for each button is set according to its label.

You can set this attribute only before the widget is realized.

EXPANDABLE attribute (Graphical interfaces only)

Indicates whether Progress extends the right-most browse-column to the right edge of the browse. This covers white space that appears when the browse is wider than the sum of the widths of the browse-columns.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget

If you set a browse's EXPANDABLE attribute to TRUE, Progress extends the right-most browse-column to the right edge of the browse, if necessary, to cover any white space that might appear-unless you explicitly set the width of the right-most browse-column using the WIDTH-CHARS or WIDTH-PIXELS attribute.

The right-most browse-column expands to the right anytime the browse or another browse-column is resized.

Notes

- If the browse has a horizontal scroll bar, no white space appears between the right-most browse column and the right edge of the browse, and the right-most browse column does not expand to the right.
- If EXPANDABLE is TRUE and a browse column's VISIBLE attribute is changed, the last column's width may be changed.
- When adding dynamic browse columns to a browse, it is best to keep EXPANDABLE turned off until all columns are added.

EXPORT( ) method (AppServer only)

Creates and modifies an AppServer's export list, which specifies the remote procedures that a client application can execute in the current AppServer session.

Return type: LOGICAL

Applies to: SESSION system handle

Syntax

```
EXPORT ( [ list ] )
```

list

A comma-separated list of procedure names and name-patterns. EXPORT( ) ignores white space (blank, tab, and newline) at the beginning and end of an entry. The only wildcard EXPORT( ) supports is the asterisk (*). For more information on wildcards, see the reference entry for the MATCHES function.

The EXPORT( ) method applies only to AppServers. That is, if the REMOTE attribute of the SESSION handle is FALSE, EXPORT( ) does nothing and returns FALSE.

The EXPORT( ) method can be called repeatedly within the context of the AppServer instance. Each time EXPORT( ) is called, the AppServer instance adds the procedures in *list* to its export list. If you do not specify *list*, the EXPORT( ) method resets the export list to empty.

The EXPORT( ) method performs pattern matching by comparing two procedure names character-by-character, taking wildcards into account. Procedure names must match exactly. Case (uppercase and lowercase) is significant.

If the EXPORT( ) method is never called, a client application can call any procedure in the AppServer's PROPATH. Once EXPORT( ) is called in the context of an AppServer, a client application can call only the procedures in the export list.

Typically, the Connect procedure or Startup procedure of an AppServer calls the EXPORT( ) method, depending on the operating mode. For example, where you might call it from the Connect procedure on a state-reset or state-aware AppServer, you would probably call it from the Startup procedure on a stateless AppServer. For more information on Connect procedures, see the reference entry for the CONNECT( ) method in this manual. For more information on both types of procedures and the AppServer, see *OpenEdge Application Server: Developing AppServer Applications*.

EXPORT-PRINCIPAL( ) method

Exports a sealed Client-principal object, with its currently defined property and attribute settings, by converting it to a raw value. You can assign this value to a RAW variable and send it to another client (such as the AppServer™). The receiving client can then import the raw value into another Client-principal object handle, using the IMPORT-PRINCIPAL( ) method, and use the imported object to set a user ID using either the SET-CLIENT( ) method or SET-DB-CLIENT function.

Return type: RAW

Applies to: Client-principal object handle

Syntax

```
EXPORT-PRINCIPAL( )
```

If the Client-principal object is in the initial (presealed) LOGOUT state, Progress generates a run-time error.

Calling this method does not generate an audit event or an audit record.

Example The following code fragment illustrates how to use the EXPORT-PRINCIPAL( ) method:

```
DEF VAR hCP as HANDLE.
DEF VAR raw-cp as RAW.
.
.
.
CREATE CLIENT-PRINCIPAL hCp.
.
.
.
raw-cp = hCP:EXPORT-PRINCIPAL( ).
```

See also IMPORT-PRINCIPAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function

EXTENT attribute

The number of elements in an array field.

Data type: INTEGER

Access: Readable

Applies to: Buffer-field object handle

Note: The EXTENT attribute corresponds to the EXTENT function.

FETCH-SELECTED-ROW( ) method

Fetches the *n*th selected row in a browse and puts the row into the database buffer. In other words, this method specifies one row from the one-based index into all currently selected rows and puts that row into the record buffer.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
FETCH-SELECTED-ROW ( n )
```

n

An integer expression that specifies a selected row within the browse.

Progress maintains a numbered list of selected rows, starting at 1. Progress builds this numbered list as the user selects rows in the browse. When you call the FETCH-SELECTED-ROW method, Progress searches this list to find the *n*th row selected by the user.

FGCOLOR attribute (Graphical interfaces only)

The color number for the foreground color of the widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

For a browse cell, it specifies the color of a specific cell in the view port. You can set this color only as the cell appears in the view port during a ROW-DISPLAY event.

Note: This attribute has no meaning for control-frames because the ActiveX control visualization constitutes the foreground.

FILE-CREATE-DATE attribute (Windows only)

Indicates the date on which the specified file was created.

Data type: DATE

Access: Readable

Applies to: FILE-INFO system handle

FILE-CREATE-TIME attribute (Windows only)

Indicates the time when the specified file was created.

Data type: INTEGER

Access: Readable

Applies to: FILE-INFO system handle

FILE-MOD-DATE attribute

Indicates the last date the specified file was modified.

Data type: DATE

Access: Readable

Applies to: FILE-INFO system handle

This attribute is supported on all platforms.

FILE-MOD-TIME attribute

Indicates the last time the specified file was modified.

Data type: INTEGER

Access: Readable

Applies to: FILE-INFO system handle

This attribute is supported on all platforms.

FILE-NAME attribute

The name of the file associated with a handle. Returns the empty string for a Web service procedure.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: COMPILER system handle, FILE-INFO system handle, RCODE-INFO system handle, THIS-PROCEDURE system handle (and all procedure handles)

The FILE-NAME attribute of the **COMPILER** handle is the name of the source file from the preceding compilation. If no error occurred during the preceding compilation, FILE-NAME assumes the Unknown value (?).

The FILE-NAME attribute of the **FILE-INFO** or **RCODE-INFO** handle is the name of the file used by subsequent references to the handle. You can specify the filename with a `.p`, `.r`, or no extension. If you set FILE-NAME to a relative pathname, Progress searches the `PROPATH` to find the file. Otherwise, Progress looks for the file specified by the absolute pathname.

The FILE-NAME attribute of a **procedure** handle is the pathname of the procedure file that contains the procedure associated with the handle. If the procedure file is specified by the Startup Procedure (-p) parameter, the attribute contains the full pathname of the file. Otherwise, it contains the pathname exactly as specified in the RUN statement that invoked it. The procedure can be local or remote. For more information on remote procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

The FILE-NAME attribute of the COMPILER handle and procedure handles is read only.

FILE-OFFSET attribute

The character offset in the source file in which a Compiler error occurred.

Data type: INTEGER

Access: Readable

Applies to: COMPILER system handle

If no error occurred in the preceding compilation, the value of FILE-OFFSET is the Unknown value (?).

FILE-SIZE attribute

Indicates the size of the specified file.

Data type: INTEGER

Access: Readable

Applies to: FILE-INFO system handle

This attribute is supported on all platforms.

FILE-TYPE attribute

A string of characters that indicate the type of file that is currently specified for the FILE-INFO handle.

Data type: CHARACTER

Access: Readable

Applies to: FILE-INFO system handle

The character string specifies two classes of file types—one type per file from the first class and one or more types per file from the second class. Table 65 lists the file type characters from the first class of file types.

Table 65: File type characters — one per file

File type	Description
D	The file is a directory.
F	The file is a standard file or FIFO pipe (UNIX systems).
M	The file is a member of a Progress procedure library.
S	The file is a special device (UNIX systems).
X	The file type is unknown. (Contact your Progress Technical Support representative if you receive this value.)

Table 66 lists the file type characters from the second class of file types.

Table 66: File type characters — one or more per file

File type	Description
H	The file is hidden.
L	The file is a symbolic link (UNIX systems).
P	The file is a pipe file (UNIX systems).
R	The file is readable.
W	The file is writeable.

FILL( ) method

Fills a ProDataSet object, recursively, based on its defined data sources, data relations, and queries. You can fill a ProDataSet object completely by starting at the top level and traversing through each of its member buffers, or partially by starting at the level of one of its member buffers.

Return type: LOGICAL

Applies to: ProDataSet object handle, Buffer object handle

Syntax

```
FILL( )
```

You can define a query for the data source of a ProDataSet member buffer at any level of the ProDataSet object to select the records to fill in one FILL operation. You can also use the FILL-WHERE-STRING attribute to override the WHERE clause in the query for the data source during a FILL operation.

You can perform a FILL operation on a ProDataSet object or one of its member buffer objects any number of times. You might do this, for example, to load data in a ProDataSet object or buffer after you have modified a Data-source object query or attached to a different Data-source object.

You can specify the FILL mode to direct the FILL operation for a ProDataSet member buffer using the FILL-MODE attribute. The default FILL-MODE is MERGE.

For more information about filling a ProDataSet object, see *OpenEdge Development: ProDataSets*.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

If Progress encounters an error, it sets the value of the ERROR attribute to TRUE for the associated ProDataSet object handle and Temp-table object handle.

See also ATTACH-DATA-SOURCE( ) method, BATCH-SIZE attribute, CREATE DATASET statement, CREATE DATA-SOURCE statement, DEFINE DATASET statement, DEFINE DATA-SOURCE statement, FILL events, FILL-MODE attribute, FILL-WHERE-STRING attribute, LAST-BATCH attribute, NEXT-ROWID attribute, RESTART-ROWID attribute, SET-CALLBACK-PROCEDURE( ) method

FILLED attribute

Indicates whether to set the background color of a rectangle to a certain value.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: RECTANGLE widget

If the value of the FILLED attribute is TRUE, the background color of the rectangle depends on the value of the BGCOLOR attribute (for graphical interfaces) or the value of the DCOLOR attribute (for character interfaces). The default value of FILLED is TRUE.

FILL-MODE attribute

Specifies the mode in which the FILL( ) method fills a ProDataSet member buffer. The default mode is MERGE.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle

Table 67 lists the FILL( ) method modes.

Table 67: FILL( ) method modes

When the mode is ...	The FILL( ) method ...
APPEND	Fills the table by adding new records on top of existing records in the table, without performing any record comparisons. If this creates duplicate records, Progress generates a run-time error and you must manage the duplicate records. If you are certain there are no duplicate records, an APPEND is more efficient than a MERGE.
EMPTY	Empties the table before the FILL operation begins.
MERGE	Fills the table by merging new records with existing records in the table. The FILL( ) method checks each record to ensure there are no duplicate records (based on the table's unique primary index). If the FILL( ) method finds a record with a duplicate key, it does not replace the record because the record might have dependent records elsewhere in the ProDataSet. In this case, Progress does not generate a run-time error. Thus, you cannot use this mode to refresh existing records.
NO-FILL	Does not perform the FILL operation on the table.
REPLACE	Fills the table by replacing existing records in the table. The FILL( ) method checks each record to determine whether or not it exists in the table (based on the table's unique primary index). If the record exists in the table, the FILL( ) method replaces it. If the record does not exist, the FILL( ) method creates a new record. A REPLACE is less efficient than an APPEND or a MERGE.

FILL-WHERE-STRING attribute

The current WHERE clause for a buffer's query, beginning with the keyword WHERE but not including the FOR EACH phrase of a prepare statement. You can use this attribute to override the WHERE clause in the query for the data-source object during a FILL operation.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Data-source object handle

See also DEFINE DATA-SOURCE statement, FILL( ) method

FIND-BY-ROWID( ) method

Locates the record with the rowid you specify, then moves the record into the buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-BY-ROWID ( rowid
  [ ,
    { SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK }
    [ , NO-WAIT ]
  ] )
```

rowid

An expression of type ROWID that represents the rowid of the desired record.

SHARE-LOCK | EXCLUSIVE-LOCK | NO-LOCK

The type of lock that Progress places on the record, if found. The default is SHARE-LOCK.

Note: For more information on record locks, see *OpenEdge Development: Progress 4GL Handbook*.

NO-WAIT

Causes FIND-BY-ROWID to return FALSE immediately if another user has a lock on the desired record and FIND-BY-ROWID specifies a locking option other than NO-LOCK.

Note: To determine whether another user has a lock on the desired record, use the LOCKED attribute of the buffer object.

The FIND-BY-ROWID method returns TRUE if it finds the record, and FALSE if it does not.

The following is an example:

```
DEFINE VARIABLE bh AS HANDLE.
DEFINE VARIABLE r AS ROWID.
r = ... /* rowid from some parameter */
bh = BUFFER CUSTOMER:HANDLE.
bh:FIND-BY-ROWID(r).
```

Note: The FIND-BY-ROWID method corresponds to a FIND statement of the form FIND *buffer* WHERE ROWID (*buffer*) = *rowid* **...**, etc. That is, triggers are honored, and the default lock mode is SHARE-LOCK. One difference, however, is that the FIND-BY-ROWID method does not raise an error if it cannot find the record.

FIND-CURRENT( ) method

Changes the lock mode of a record in a buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-CURRENT ( [ lockmode [ , wait-mode ] ] )
```

lockmode

An integer expression evaluating to one of the following constants: SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK. You can assign any of these constants to an integer variable. For example, mylock = NO-LOCK. The default is SHARE-LOCK.

waitmode

An integer expression evaluating to one of the following: NO-WAIT, 0, or the Unknown value (?). You can assign NO-WAIT to an integer variable. For example, mywait = NO-WAIT.

The default is to wait.

The following shows an example of the FIND-CURRENT method:

```
DEFINE VARIABLE bh AS HANDLE.

bh = BUFFER customer:HANDLE.

DO Transaction:
bh:FIND-CURRENT(EXCLUSIVE-LOCK).
END.
MESSAGE bh:CURRENT-CHANGED cust.cust-num.
```

If the change in lock status succeeds, the method returns TRUE, otherwise it returns FALSE.

If the lock change fails, a message displays. You can suppress the message using NO-ERROR on the statement containing the method.

Executing the FIND-CURRENT method resets the CURRENT-CHANGED attribute. If the record in the database changes between the time the original record was found and the FIND-CURRENT executes, the CURRENT-CHANGED attribute returns TRUE. If the record does not change, then the CURRENT-CHANGED attribute returns FALSE.

See also FIND-BY-ROWID( ) method, CURRENT-CHANGED attribute, CURRENT-CHANGED function, FIND-FIRST( ) method, FIND-LAST( ) method, FIND-UNIQUE( ) method, FIND statement

FIND-FIRST( ) method

Gets a single record. This method lets a user get the first record that satisfies the predicate expression.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-FIRST (predicate-expression [ , lockmode [ , wait-mode ] ] )
```

predicate-expression

A character expression that evaluates to the following syntax:

```
[ WHERE [ logical-expression ] ] [ USE-INDEX index-name ]
```

Once evaluated, *predicate-expression* can contain only constants and unabbreviated references to fields from the buffer.

The *predicate-expression* itself can be built using a concatenation of character expressions.

lockmode

An integer expression evaluating to one of the following constants: SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK. You can assign any of these constants to an integer variable. For example, mylock = NO-LOCK.

The default is SHARE-LOCK.

wait-mode

An integer expression evaluating to one of the following: NO-WAIT, 0, or the Unknown value (?). You can assign NO-WAIT to an integer variable. For example, mywait = NO-WAIT. The default is to wait.

The following shows some examples of FIND-FIRST method:

```
DEFINE VARIABLE bh AS HANDLE.
DEFINE VARIABLE myname AS char.

bh = BUFFER customer:HANDLE.

bh:FIND-FIRST("where cust-num > 2", NO-LOCK).
do transaction ;
  bh:FIND-FIRST("", EXCLUSIVE-LOCK).
end.
bh:FIND-FIRST("where name = " + QUOTER(myname) , NO-LOCK).
```

If FIND-FIRST succeeds, it returns TRUE, otherwise it returns FALSE.

If FIND-FIRST fails, it does not raise an error but displays a message. You can suppress this message by using NO-ERROR on the statement containing the method.

See also FIND-BY-ROWID( ) method, FIND-CURRENT( ) method, FIND-LAST( ) method, FIND-UNIQUE( ) method, FIND statement

FIND-LAST( ) method

Gets a single record. This method lets a user get the last record that satisfies the predicate expression.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-LAST  ( predicate-expression [ , lockmode [ , wait-mode ] ] )
```

predicate-expression

A character expression that evaluates to the following syntax:

```
[ WHERE [  logical-expression ] ] [ USE-INDEX index-name ]
```

Once evaluated, *predicate-expression* can contain only constants and unabbreviated references to fields from the buffer.

The *predicate-expression* itself can be built using a concatenation of character expressions.

lockmode

An integer expression evaluating to one of the following constants: SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK. You can assign any of these constants to an integer variable. For example, mylock = NO-LOCK.

The default is SHARE-LOCK.

wait-mode

An integer expression evaluating to one of the following: NO-WAIT, 0, or the Unknown value (?). You can assign NO-WAIT to an integer variable. For example, mywait = NO-WAIT. The default is to wait.

The following shows some examples of the FIND-LAST method:

```
DEFINE VARIABLE bh AS HANDLE.

bh = BUFFER customer:HANDLE.

bh:FIND-LAST("where balance > 0 use-index name").
bh:FIND-LAST("where cust-num > 5 and address < 'z'").
```

If FIND-LAST succeeds, it returns TRUE, otherwise it returns FALSE.

If FIND-LAST fails, it does not raise an error but displays an error message. You can suppress the message using NO-ERROR on the statement containing the method.

See also FIND-CURRENT( ) method , FIND-FIRST( ) method, FIND-UNIQUE( ) method, FIND statement

FIND-UNIQUE( ) method

Gets a single record. This method lets a user get a unique record that satisfies the predicate expression.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
FIND-UNIQUE ( predicate-expression [ , lockmode [ , wait-mode ] ] )
```

predicate-expression

A character expression that evaluates to the following syntax:

```
[ WHERE [ logical-expression ] ] [ USE-INDEX index-name ]
```

Once evaluated, *predicate-expression* can contain only constants and unabbreviated references to fields from the buffer. The *predicate-expression* itself can be built using a concatenation of character expressions.

lockmode

An integer expression evaluating to one of the following constants: SHARE-LOCK, EXCLUSIVE-LOCK, or NO-LOCK. You can assign NO-LOCK to an integer variable. For example, mylock = NO-LOCK.

The default is SHARE-LOCK.

waitmode

An integer expression evaluating to one of the following: NO-WAIT, 0, or the Unknown value (?). The default is to wait.

You can assign NO-WAIT to an integer variable. For example, mywait = NO-WAIT.

The following shows some examples of the FIND-UNIQUE method:

```
DEFINE VARIABLE bh AS HANDLE.
DEFINE VAR myname as char

bh = BUFFER customer:HANDLE.


bh:FIND-UNIQUE("where cust-num < 3 and name = 'lift line skiing'").

bh:FIND-UNIQUE("where cust-num = 30").

bh:FIND-UNIQUE("where name = " + QUOTER(myname).
MESSAGE bh:AMBIGUOUS.
```

If FIND-UNIQUE succeeds, it returns TRUE, otherwise it returns FALSE.

If FIND-UNIQUE fails, a message displays. You can suppress this message using NO-ERROR on the statement containing the method.

If more than one record satisfies the predicate expression, then the AMBIGUOUS attribute is set to TRUE.

See also FIND-FIRST( ) method, FIND-CURRENT( ) method, FIND-LAST( ) method, FIND statement

FIRST-ASYNC-REQUEST attribute

Returns the first entry in the list of all current asynchronous request handles for the specified AppServer or Web service that have been created in the current session.

Data type: HANDLE

Access: Readable

Applies to: Server object handle

If there are no asynchronous request handles for the specified server, FIRST-ASYNC-REQUEST returns the Unknown value (?).

FIRST-BUFFER attribute

Returns the buffer handle of the first table that has a buffer object. The table may be either a temp-table or a connected database, in that order. If no temp-table or database buffers exist in the session, it returns the Unknown value (?).

Data type: WIDGET-HANDLE

Access: Readable

Applies to: SESSION system handle

There is no LAST-BUFFER attribute associated with the SESSION handle since the chain is one-directional.

FIRST-CHILD attribute

The handle of the first widget created in the container widget or the current session.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: DIALOG-BOX widget, FIELD-GROUP widget, FRAME widget, MENU widget, SUB-MENU widget, WINDOW widget, SESSION system handle

You can use the FIRST-CHILD attribute to find the first entry in a list of all frames and dialog boxes in a window, all field groups in a frame or dialog box, all widgets in a field group, all menu items in a menu or submenu, or all windows in an OpenEdge session. After finding the first entry, you can find the remaining entries in the list by using each widget's NEXT-SIBLING attribute.

FIRST-COLUMN attribute

A handle to the first column in a browse widget, regardless of the value of its READ-ONLY attribute or its VISIBLE attribute.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: BROWSE widget

After finding the first column, accessing the NEXT-COLUMN attribute of the current column allows you to walk through the browse columns.

FIRST-DATASET attribute

A handle to the first dynamic ProDataSet object created in the current OpenEdge session.

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

After finding the first entry, you can find the remaining entries in the list by using the NEXT-SIBLING attribute for each dynamic ProDataSet object. Use the NEXT-SIBLING attribute to get the next entry in the list of ProDataSet object handles created in the current OpenEdge session.

FIRST-DATA-SOURCE attribute

A handle to the first dynamic Data-source object created in the current OpenEdge session.

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

Use the NEXT-SIBLING attribute to get the next entry in the chain of dynamic Data-source object handles created in the current OpenEdge session.

FIRST-OBJECT attribute

The object reference for the first class object instance in the list of all valid instances created in the current OpenEdge session. If there are no class object instances in the current session, this attribute returns the Unknown value (?).

Data type: Progress.Lang.Object

Access: Readable

Applies to: SESSION system handle

Once you get the first object reference in the list, you can use the NEXT-SIBLING data member in the Progress.Lang.Object class to get the next entry in the list of object references.

See also LAST-OBJECT attribute, NEXT-SIBLING and PREV-SIBLING data members (in the Progress.Lang.Object class)

FIRST-PROCEDURE attribute

For AppServer, returns the first entry in the list of remote persistent procedures running on the connected AppServer. For Web services, returns the first entry in the list of procedure objects associated with the Web service.

Data type: HANDLE

Access: Readable

Applies to: Server object handle, SESSION system handle

If the current session has no active persistent procedures or the AppServer has no active remote persistent procedures, FIRST-PROCEDURE has the Unknown value (?). To find the next persistent procedure given the first, use the NEXT-SIBLING attribute of the procedure handle.

For information on creating persistent procedures, see the RUN statement reference entry in this book. For more information on the AppServer, see *OpenEdge Application Server: Developing AppServer Applications*. To check a handle for validity, use the VALID-HANDLE function.

FIRST-QUERY attribute

A handle to the first dynamic query created in the current OpenEdge session.

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

Use the NEXT-SIBLING attribute to get the next entry in the chain of dynamic query handles created in the current OpenEdge session. The chain of dynamic query handles includes all automatically generated queries, such as those created for data-relation objects.

FIRST-SERVER attribute

A handle to the first entry in the list of server handles of the current OpenEdge session. This includes both AppServer server objects and Web service server objects.

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

The handle associated with the first entry in the list of all server handles created in the current session. If the current session has no server handles, FIRST-SERVER has the Unknown value (?). For more information on server handles, see the Server object handle reference entry.

FIRST-SERVER-SOCKET attribute

A handle to the first entry in the list of all valid server socket handles created in the current session. If there are no server socket handles in this session, FIRST-SERVER-SOCKET returns the Unknown value (?).

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

FIRST-SOCKET attribute

A handle to the first entry in the list of all valid socket handles created in the current session. If there are no socket handles in this session, FIRST-SOCKET returns the Unknown value (?).

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

FIRST-TAB-ITEM attribute

The first widget in the tab order of a field group.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: FIELD-GROUP widget

When you set this attribute, the assigned widget is moved to the first tab position, preceding the widget that was previously at this position. Other widgets in the field group maintain their same relative tab positions.

To set the attribute, you must assign it the widget handle of a field-level widget or frame that can receive focus from a TAB event and that is also a child of the field group to which the attribute applies. If the FIRST-TAB-ITEM attribute is not set (that is, is the Unknown value (?)), the default first tab position goes to the widget identified by the FIRST-CHILD attribute of the field group.

For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry in this manual and the chapter on frames in *OpenEdge Development: Progress 4GL Handbook*.

Note: Any tab reordering that you do with this attribute can be reset by a subsequent ENABLE statement unless you define the frame that owns the field group with the KEEP-TAB-ORDER option. For more information, see the ENABLE statement and Frame phrase reference entries.

FIT-LAST-COLUMN attribute (Graphical interfaces only)

Allows the browse to be displayed so that there is no empty space to the right and no horizontal scroll bar by potentially widening or shrinking the last browse column's width.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

When this attribute is specified, and the last browse column can be fully or partially displayed in the browse's viewport, then the last browse column's width is adjusted so that it fits within the viewport with no empty space to its right and no horizontal scroll bar.

If the last browse column is fully contained in the viewport with empty space to its right, it grows so that its right edge is adjacent to the vertical scroll bar.

If the last browse column extends outside the viewport, it shrinks so its right edge is adjacent to the vertical scroll bar and the horizontal scroll bar is not needed.

The default value is FALSE.

Note: The FIT-LAST-COLUMN attribute performs the same function as the EXPANDABLE attribute. Progress Software recommends that you use the FIT-LAST-COLUMN attribute instead of the EXPANDABLE attribute. This recommendation includes replacing EXPANDABLE with FIT-LAST-COLUMN in your current code.

The following shows the DEFINE BROWSE statement syntax with FIT-LAST-COLUMN specified:

```
DEFINE BROWSE b1 QUERY q1
DISPLAY customer.cust-num customer.name
ENABLE customer.cust-num WITH  3 DOWN WIDTH 40 FIT-LAST-COLUMN
```

The MIN-COLUMN-WIDTH attribute affects the FIT-LAST-COLUMN attribute. As a result, if FIT-LAST-COLUMN is set to TRUE, the last browse column is resized to fit within the viewport only if its width is no smaller than the minimum width. To specify the minimum size that the last browse column's width can be reduced to, use the MIN-COLUMN-WIDTH-PIXELS or MIN-COLUMN-WIDTH-CHARS attribute. See MIN-COLUMN-WIDTH-PIXELS attribute and MIN-COLUMN-WIDTH-CHARS attribute for more information.

FIT-LAST-COLUMN and NO-EMPTY-SPACE are mutually exclusive. If both are specified in the DEFINE BROWSE statement, the compiler displays an error message. If one attribute is set to TRUE while the other attribute is already TRUE, a warning message displays at run time.

FIT-LAST-COLUMN is primarily intended for use in the initial layout of a static browse. It is most useful when laying out a browse with a specified width when you have only a few browse columns and you want to fully use the available space in your viewport.

If the FIT-LAST-COLUMN attribute is set to TRUE, and, subsequently, any browse column's width is changed or the browse's width is changed, then the last browse column's width might be adjusted so that it fits within the viewport with no empty space and no horizontal scroll bar.

When the last browse column's width is set at run time after the browse is realized, then FIT-LAST-COLUMN is ignored.

If the FIT-LAST-COLUMN attribute is set to FALSE, the last browse column's width remains the same and is never changed by Progress.

The FIT-LAST-COLUMN attribute and the EXPANDABLE attribute have the same behavior. Therefore, if you specify the Expand Browse (`-expandbrow`) startup parameter at startup, the FIT-LAST-COLUMN attribute is set to TRUE for each browse in that session.

See also DEFINE BROWSE statement

FLAT-BUTTON attribute (Windows only; Graphical interfaces only)

Indicates whether a button is two-dimensional until the mouse passes over it, at which time, a 3D border appears.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BUTTON widget

The FLAT-BUTTON attribute must be set before the button is realized. The default value is FALSE.

Setting the FLAT-BUTTON attribute to TRUE forces the NO-FOCUS attribute to TRUE because the FLAT-BUTTON attribute only works with the NO-FOCUS attribute. Similarly, setting the NO-FOCUS attribute to FALSE forces the FLAT-BUTTON attribute to FALSE.

The mnemonic key (**ALT** accelerator) for a widget will not work if the NO-FOCUS attribute is TRUE because this removes the widget from the tab order. Also, because the widget is not in the tab order, pressing **TAB** will not change focus from the widget.

FOCUSED-ROW attribute

The 1-based index or position of the focused row in the viewport.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget

FOCUSED-ROW-SELECTED attribute

Indicates whether the row that has focus is selected.

Data type: LOGICAL

Access: Readable

Applies to: BROWSE widget

If the row that has focus is selected, FOCUSED-ROW-SELECTED is TRUE . Otherwise, it is FALSE.

FONT attribute (Graphical interfaces only)

The number of the font of a widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, LITERAL widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The font number represents an entry in the font table maintained by the FONT-TABLE handle.

For a browse cell, it specifies the font of a specific cell in the view port. You can set this font only as the cell appears in the view port during a ROW-DISPLAY event.

Note: When the AUTO-RESIZE attribute is set to TRUE, Progress resizes the following widgets with run-time changes to the FONT attribute: Buttons, Combo boxes, Editors, Fill-ins, Radio sets, Selection lists, Sliders, Texts, Toggle boxes

FOREGROUND attribute

Indicates whether the field group is a foreground or a background field group.

Data type: LOGICAL

Access: Readable

Applies to: FIELD-GROUP widget

If the FOREGROUND attribute is TRUE, the field group is a foreground (data iteration) group. If FOREGROUND is FALSE, the field group is the background group for the frame.

FORM-INPUT attribute

Returns raw HTTP form input that is less than 32K in size. Do not access this attribute.

Data type: CHARACTER

Access: Readable

Applies to: WEB-CONTEXT system handle

FORM-LONG-INPUT attribute

Returns raw HTTP form input that is greater than 32K in size. Progress performs no conversion on the data. Do not access this attribute.

Data type: MEMPTR

Access: Readable

Applies to: WEB-CONTEXT system handle

FORMAT attribute

The text format of a widget or browse-cell.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (cell, Windows only), Buffer-field object handle, COMBO-BOX widget, FILL-IN widget, TEXT widget, TOGGLE-BOX widget

For DROP-DOWN-LIST combo-boxes, if you set this attribute with items in the drop-down list, all items are converted to the new format. This attribute is ignored for SIMPLE and DROP-DOWN combo-boxes.

For combo boxes whose entries consist of label-value pairs, Progress converts all values to the new format.

For browses in Windows, if you modify the FORMAT attribute of a browse-cell, its format changes, but its size does not.

For buffer-fields, the value of the FORMAT attribute does not affect the Progress user interface anywhere. Rather, it controls the output of the STRING-VALUE attribute, and lets users explicitly format non-Progress user interfaces.

Note: When the AUTO-RESIZE attribute is TRUE, Progress resizes combo box and fill-in field widgets with run-time changes to the FORMAT attribute.

For information on text formats, see *OpenEdge Development: Progress 4GL Handbook*.

FORMATTED attribute

Determines the format of XML output from a SAX-writer object.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SAX-writer object handle

The default value is FALSE.

TRUE indicates that the SAX-writer should format the document with additional white space, carriage returns, and line feeds, so that the elements display in a hierarchical manner.

FALSE indicates the SAX-writer should create an optimized document that includes no extra white space.

You can read this attribute at all times, but you can only write to it when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, you can only change the attribute when the SAX-writer is not writing, otherwise the call fails and generate an error message.

FORWARD-ONLY attribute

Lets you avoid building result-lists for static and dynamic queries.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Query object handle

Set to TRUE to avoid building result-lists for queries. Set to FALSE to build result-lists for queries. The default is FALSE.

When TRUE, you cannot use the GET PREV, GET LAST, REPOSITION, or BROWSE methods or statements with these queries. If you do, OpenEdge generates an error. You can use the GET-FIRST( ) method and GET FIRST statement only on newly opened queries, and you can use the GET NEXT statement and GET-NEXT( ) method freely.

If you set FORWARD-ONLY to TRUE, and you open a query with preselect or sort, Progress still builds a result-list in order to resolve the query. You cannot set FORWARD-ONLY while a query is open or being browsed.

Setting FORWARD-ONLY to TRUE can improve the performance of operations on queries.

FRAGMENT attribute

Specifies if the output of a SAX-writer object is a complete document or a fragment.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SAX-writer object handle

The default value is FALSE.

TRUE indicates that the writer should not include the XML declaration or require a root node. This behavior allows the developer to create XML fragments which can be used to create larger documents. For example, one SAX-writer object's document fragment LONGCHAR could be used as the parameter of WRITE-FRAGMENT for another SAX-writer.

FALSE indicates the SAX-writer should create a complete XML document with the XML declaration and root node.

You can read this attribute at all times, but you can only write to it when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, the attribute can only be changed when the SAX-writer is not writing, otherwise it fails and generates an error message.

FRAME attribute

The handle of the frame that contains the widget.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

This attribute is writeable only for static frames and all dynamic widgets. You can set this attribute for a static frame only before the widget is realized.

FRAME-COL attribute

The decimal column position, in character units, of the left edge of the widget relative to the upper left corner of the frame that contains the widget.

Data type: DECIMAL

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

FRAME-NAME attribute

The name of the frame that contains the widget.

Data type: CHARACTER

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

FRAME-ROW attribute

The decimal row position, in character units, of the top edge of the widget relative to the upper left corner of the frame that contains the widget.

Data type: DECIMAL

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

If the parent frame is a down frame with multiple occurrences, the FRAME-ROW attribute regards the original occurrence as the parent, not the current occurrence.

FRAME-SPACING attribute

The number of display units between frames in a window. In graphical interfaces the display units are pixels. In character interfaces the display units are character cells.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

By default, the value for FRAME-SPACING is the height of one row in the default system font. In character interfaces, this is the character cell height. In graphical interfaces, this is the number of pixels returned by the PIXELS-PER-ROW attribute.

FRAME-X attribute

The location of the left edge of the widget relative to the upper left corner of the frame that contains the widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

In character mode, this attribute returns the widget location in row column units. In graphical interfaces, this attribute returns pixels.

FRAME-Y attribute

The location of the top edge of the widget relative to the upper left corner of the frame that contains the widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

In character mode, this attribute returns the widget location in row column units. In graphical interfaces, this attribute returns pixels.

FREQUENCY attribute

Indicates the incremental display of the TIC-MARKS attribute. It is used exclusively with the TIC-MARKS attribute.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SLIDER widget

For example, if you set FREQUENCY to 5, a tic mark appears in every fifth position along the length of the slider.

FULL-HEIGHT-CHARS attribute

The maximum internal height of the window, in character units.

Data type: DECIMAL

Access: Readable

Applies to: WINDOW widget

The maximum internal height of a window is the height of the screen display minus the vertical spacing required to display the border, title bar, menu bar, message area, and status area of the window.

The value of this attribute is the Unknown value (?) until the window is realized.

FULL-HEIGHT-PIXELS attribute

The maximum internal height of the window, in pixel units.

Data type: INTEGER

Access: Readable

Applies to: WINDOW widget

The maximum internal height of a window is the height of the screen display minus the vertical spacing required to display the border, title bar, menu bar, message area, and status area of the window.

The value of this attribute is the Unknown value (?) until the window is realized.

FULL-PATHNAME attribute

The absolute pathname of the file specified in the FILE-NAME attribute.

Data type: CHARACTER

Access: Readable

Applies to: FILE-INFO system handle

FULL-WIDTH-CHARS attribute

The maximum internal width of the window, in character units.

Data type: DECIMAL

Access: Readable

Applies to: WINDOW widget

The maximum internal width of a window is the width of the screen display minus the horizontal spacing required to display the border of the window.

The value of this attribute is the Unknown value (?) until the window is realized.

FULL-WIDTH-PIXELS attribute

The maximum internal width of the window, in pixel units.

Data type: INTEGER

Access: Readable

Applies to: WINDOW widget

The maximum internal width of a window is the width of the screen display minus the horizontal spacing required to display the border of the window.

The value of this attribute is the Unknown value (?) until the window is realized.

FUNCTION attribute

The names of high-level events based on the EVENT-TYPE attribute value.

Data type: CHARACTER

Access: Readable

Applies to: LAST-EVENT system handle

For EVENT-TYPE = "KEYPRESS", this attribute returns key functions, such as "RETURN". For EVENT-TYPE = "MOUSE", this attribute returns high-level events for both portable and three-button event types, such as "MOUSE-SELECT-CLICK" (portable) or "LEFT-MOUSE-CLICK" (three-button). For EVENT-TYPE = "PROGRESS", this attribute returns high-level widget and direct manipulation events, such as "CHOOSE" or "SELECTION".

GET-ATTRIBUTE( ) method

Returns the value of the specified attribute of an element referred to by an XML node reference.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
GET-ATTRIBUTE( name )
```

name

The attribute name whose value is desired. Attribute names are defined within the element tag. If using a DTD, you can define attributes with the "IMPLIED" property and those attributes will appear in the DOM structure.

If `hNoderef` is an element node with various attributes, and `anames` and `bname` are character program variables, the following example demonstrates listing all the attributes of the node:

```
anames = hNoderef:ATTRIBUTE-NAMES.
REPEAT j = 1 TO NUM-ENTRIES(anames):
  bname = ENTRY(j,anames).
  MESSAGE "attribute-name is" bname "value is"
        hNoderef:GET-ATTRIBUTE(bname).
END.
```

GET-ATTRIBUTE-NODE( ) method

Returns the XML ATTRIBUTE node with the specified name.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
GET-ATTRIBUTE-NODE( attr-node-handle, name )
```

attr-node-handle

A valid X-noderef handle to use for the XML ATTRIBUTE node.

name

A character expression representing the name of the XML ATTRIBUTE node. For a namespace-aware ATTRIBUTE node, you must qualify the node name with a prefix including a colon (for example, *prefix:node-name*).

GET-BINARY-DATA( ) method

Returns a MEMPTR containing the binary data in the file specified in the form field. Progress sets the size of the MEMPTR to match the size of the file. This method is called by the `get-binary-data` WebSpeed API function. Intended for internal use only.

Return type: MEMPTR

Applies to: WEB-CONTEXT system handle

Syntax

```
GET-BINARY-DATA (INPUT field-name)
```

field-name

The name of the form field containing the name of the file posted in the web request received by the WebSpeed Agent.

If the specified field is not part of the form, or the field is not of type ‘file’, Progress returns the Unknown value (?) and displays an error message. You can suppress this message by using NO-ERROR on the statement containing the method.

GET-BLUE-VALUE( ) method (Graphical interfaces only)

Returns the blue component of an entry in the color table.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-BLUE-VALUE ( index )
```

index

An integer expression that specifies an entry in the color table.

GET-BROWSE-COLUMN( ) method

Returns the widget-handle for the requested browse column.

Return type: WIDGET-HANDLE

Applies to: BROWSE widget

Syntax

```
GET-BROWSE-COLUMN( col-index )
```

col-index

An integer value specifying the 1-based index into the browse column list.

GET-BUFFER-HANDLE( ) method

Gets the handle to a particular buffer of a query or ProDataSet object.

Return type: HANDLE

Applies to: ProDataSet object handle, Query object handle

Syntax

```
GET-BUFFER-HANDLE ( buffer-sequence-number | buffer-name )
```

buffer-sequence-number

An INTEGER that represents the sequence number of the desired buffer.

Note: Sequence numbers for buffers of a query start at one, where one represents the top level and subsequent numbers represent lower levels of join, if any.

buffer-name

A CHARACTER expression that evaluates to the name of a buffer in the query or ProDataSet object.

If the buffer cannot be found, this attribute returns the Unknown value (?).

GET-BYTES-AVAILABLE( ) method

Indicates the number of bytes available for reading from the socket.

Return type: INTEGER

Applies to: Socket object handle

Syntax

```
GET-BYTES-AVAILABLE( )
```

GET-CALLBACK-PROC-CONTEXT( ) method

Returns the handle of the procedure that contains the internal procedure associated with the Progress callback for the specified event.

Return type: HANDLE

Applies to: Buffer object handle, ProDataSet object handle, Query object handle

Syntax

```
GET-CALLBACK-PROC-CONTEXT( event-name )
```

event-name

The name of a defined event.

If the object does not have a callback procedure for the specified event, this method returns the Unknown value (?).

Use the SET-CALLBACK-PROCEDURE( ) method to associate an internal procedure with a callback for an object.

For more information on events, see the "Events Reference" section on page 2171.

See also APPLY-CALLBACK( ) method, GET-CALLBACK-PROC-NAME( ) method, SET-CALLBACK-PROCEDURE( ) method

GET-CALLBACK-PROC-NAME( ) method

Returns the name of the internal procedure associated with the Progress callback for the specified event.

Return type: CHARACTER

Applies to: Buffer object handle, ProDataSet object handle, Query object handle

Syntax

```
GET-CALLBACK-PROC-NAME( event-name )
```

event-name

The name of a defined event.

If the object does not have a callback procedure for the specified event, this method returns the Unknown value (?).

Use the SET-CALLBACK-PROCEDURE( ) method to associate an internal procedure with a callback for an object.

For more information on events, see the "Events Reference" section on page 2171.

See also APPLY-CALLBACK( ) method, GET-CALLBACK-PROC-CONTEXT( ) method, SET-CALLBACK-PROCEDURE( ) method

GET-CGI-LIST( ) method

Gets the list of CGI environment variables. This method is called by the `get-cgi` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

GET-CGI-VALUE( ) method

Gets the value of a specified CGI environment variable. This method is called by the `get-cgi` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

GET-CGI-LONG-VALUE( ) method

Returns a LONGCHAR value in either the code page specified in the HTML-CHARSET attribute, if that code page is valid for a LONGCHAR, or `-cpinternal`. Otherwise, it returns the Unknown value (`?`) and displays an error message. You can suppress this message by using NO-ERROR on the statement containing the method. This method is called by the `get-cgi-long` and `get-long-value` WebSpeed API functions. Intended for internal use only.

Return type: LONGCHAR

Applies to: WEB-CONTEXT system handle

GET-CHANGES( ) method

Loads an empty ProDataSet object with changed rows from either a single temp-table or all temp-tables in another ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle

Syntax

```
change-handle:GET-CHANGES(original-handle [, get-parent-mode ] )
```

change-handle

A handle to the ProDataSet object or ProDataSet temp-table buffer to receive the changed rows.

original-handle

A handle to the source ProDataSet object or ProDataSet temp-table buffer that contains the changed rows.

get-parent-mode

An optional logical expression where TRUE indicates that Progress get changed rows in a *get-parent* mode.

When TRUE, Progress includes the parent row of each changed child row in the ProDataSet object or ProDataSet temp-table (if any). If there is more than one parent level above the changed row, Progress includes the parent row at each level. In this case, the parent temp-tables must have a unique primary index that Progress can use to find the corresponding rows. If a parent row has changed, Progress copies both the before-image and after-image of the parent row. If a parent row has not changed, there will be no before-image of the parent row, and its change state (ROW-STATE) will be ROW-UNMODIFIED (0) or the Unknown value (?).

Note: When the relation mode of a parent is REPOSITION, no attempt is made to find that parent.

When FALSE, Progress does not include parent rows. The default value is FALSE.

The ProDataSet objects associated with the *change-handle* and *original-handle* must have the same number of temp-table buffers, and the definition of the corresponding temp-tables must match (that is, in the number of columns, data types, and so on).

Once the changed rows are loaded, Progress sets the ORIGIN-HANDLE attribute on the temp-tables in the receiving ProDataSet object to the corresponding temp-tables in the original source ProDataSet object. Progress also sets the ORIGIN-ROWID attribute on each of the before-image table rows created in the receiving ProDataSet object to the ROWID of the corresponding before-image table row in the original source temp-table. The MERGE-CHANGES( ) method and MERGE-ROW-CHANGES( ) method uses these values to match up temp-tables and temp-table rows during a merge operation.

GET-CHILD( ) method

Retrieves a specific child node of the current node. The first parameter must be a valid X-noderef handle and will refer to the specified child XML node if the method succeeds.

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
GET-CHILD( x-node-handle , index )
```

x-node-handle

A valid X-noderef handle to use as the child XML node.

index

An integer representing the relative number in the node-tree (1 based).

The following code fragment demonstrates getting all the child nodes from the XML node referenced by hNoderef using the GET-CHILD( ) method:

```
. . .
REPEAT j = 1 TO hNoderef:NUM-CHILDREN:
  ok = hNoderef:GET-CHILD(hNoderefChild,j).
  IF NOT ok THEN LEAVE.
   . . .
END.
```

GET-CHILD-RELATION( ) method

Gets the handle to a data-relation object for which the buffer is the parent.

Return type: HANDLE

Applies to: Buffer object handle

Syntax

```
GET-CHILD-RELATION ( index )
```

index

An integer expression indicating the 1-based index of the data-relation object.

GET-CONFIG-VALUE( ) method

Gets the value of parameters set in the WebSpeed configuration file. This method is called by the `get-config` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

GET-CURRENT( ) method

Refetches the current record or records associated with the query.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-CURRENT ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
  | EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-DATASET-BUFFER( ) method

Gets the handle to the ProDataSet object buffer associated with the Data-source object.

Return type: HANDLE

Applies to: Data-source object handle

Syntax

```
GET-DATASET-BUFFER ( )
```

GET-DOCUMENT-ELEMENT( ) method

Retrieves the root element of the document. The parameter must be a valid X-noderef handle and will refer to the document's root element if the method succeeds.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
GET-DOCUMENT-ELEMENT( x-node-handle )
```

x-node-handle

A valid X-noderef handle to use for the root element.

The following example demonstrates the use of GET-DOCUMENT-ELEMENT if hDoc is an X-document and hRoot is an X-noderef:

```
/* Creates a 4GL document object & initializes the associated XML object. */
CREATE X-document hDoc.

/* Creates a 4GL reference for an XML node in a parse tree. */
CREATE X-NODEREF hRoot.

/* Reads the myxml.xml document into an XML parse tree. */
hDoc:LOAD("file","myxml.xml",false).

/* Associates hRoot with the root node of the hDoc document. */
hDoc:GET-DOCUMENT-ELEMENT(hRoot).
```

GET-DROPPED-FILE( ) method (Windows only; Graphical interfaces only)

Returns the name of the dropped file indicated by the *index* parameter.

Return type: CHARACTER

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
GET-DROPPED-FILE ( index )
```

If *index* is less than 1 or greater than the number of dropped files (as indicated by the NUM-DROPPED-FILES attribute), GET-DROPPED-FILE( ) returns the Unknown value (?). If there is no current drag-and-drop operation, GET-DROPPED-FILE( ) returns the Unknown value (?).

GET-DYNAMIC( ) method (Graphical interfaces only)

Returns TRUE if the entry in the color table is a dynamic color.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-DYNAMIC ( index )
```

index

An integer expression that specifies an entry in the color table.

GET-FIRST( ) method

Moves a query object's result list pointer to the first row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-FIRST ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
  | EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-GREEN-VALUE( ) method (Graphical interfaces only)

Returns the green component of an entry in the color table.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-GREEN-VALUE ( index )
```

index

An integer expression that specifies an entry in the color table.

GET-HEADER-ENTRY( ) method

Retrieves the SOAP-header-entryref object at the given header entry index (base 1).

Return type: LOGICAL

Applies to: SOAP-header object handle

Syntax

```
GET-HEADER-ENTRY( header-entryref, index )
```

header-entryref

A valid SOAP-header-entryref object handle.

index

An integer expression indicating the 1-based index of the header entry.

GET-INDEX-BY-NAMESPACE-NAME( ) method

Gets the 1-based index of the attribute with the given namespace name.

Return type: INTEGER

Applies to: SAX-attributes object handle

Syntax

```
GET-INDEX-BY-NAMESPACE-NAME ( uri, localname )
```

uri

The namespace URI (that is, the URI to which the attribute's prefix refers), or if the name has no namespace URI, an empty string.

localname

The local (unqualified) name of the attribute.

Returns the Unknown value (?) if no attribute's uri-localname combination matches the combination of *uri* and *localname*.

GET-INDEX-BY-QNAME( ) method

Gets the 1-based index of the attribute with the given XML qualified name.

Return type: INTEGER

Applies to: SAX-attributes object handle

Syntax

```
GET-INDEX-BY-QNAME ( qname )
```

qname

A CHARACTER expression indicating the XML qualified name of the attribute of interest.

Returns the Unknown value (?) if no attribute's XML qualified name matches *qname*.

GET-ITERATION( ) method

Returns the widget handle for the field group that represents the *n*th visible iteration of the frame.

Return type: WIDGET-HANDLE

Applies to: FRAME widget

Syntax

```
GET-ITERATION ( n )
```

n

An integer expression that specifies the number of a visible frame iteration.

You can read the NUM-ITERATIONS attribute of the frame to determine how many visible foreground (data) iterations the frame contains. You can then use the FIRST-CHILD or LAST-CHILD attributes of the field group to access the first or last field-level widget (respectively) in the iteration.

GET-LAST( ) method

Moves a query object's result list pointer to the last row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-LAST ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
  | EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-LOCALNAME-BY-INDEX( ) method

Gets the local (unqualified) name of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-LOCALNAME-BY-INDEX ( index )
```

index

An integer expression indicating the 1-based index of the attribute.

Looks up an attribute's local (unqualified) name by index. Returns the Unknown value (?) if namespace processing is disabled, if the index is less than 1, or if the index is greater than the value of the SAX-attributes object's NUM-ITEMS attribute.

GET-MESSAGE( ) method

Returns the error message associated with the *n*th error that occurred during the execution of a statement run with the NO-ERROR option.

Return type: CHARACTER

Applies to: ERROR-STATUS system handle

Syntax

```
GET-MESSAGE ( n )
```

n

An integer expression that specifies the error whose message you want to retrieve.

GET-NEXT( ) method

Moves a query object's result list pointer ahead one row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-NEXT ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
  | EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-NODE( ) method

Returns a handle to an X-noderef that refers to the XML underlying a SOAP header entry or SOAP fault entry.

Return type: LOGICAL

Applies to: SOAP-fault-detail object handle, SOAP-header-entryref object handle

Syntax

```
GET-NODE ( x-noderef )
```

x-noderef

A variable of type X-noderef that refers to the root node of a DOM tree that has a SOAP header entry or SOAP fault entry as its root.

The X-noderef will have namespace declarations for all namespaces that are in effect for the SOAP header entry element. The X-noderef will include attributes for all attributes that the SOAP header entry has, including SOAP ENV:mustUnderstand and SOAP-ENV:Actor. Operations performed on the X NODEREF, its child X-noderefs, will directly affect the underlying header entry (note that this contrasts with the LONGCHAR returned from GET-SERIALIZED( ) method).

Returns the Unknown value (?) if the SOAP-header-entryref object handle has been initialized but does not refer to a header entry (for example, immediately after the CREATE SOAP-HEADER-ENTRYREF statement.)

GET-NUMBER( ) method

Returns the error number associated with the *n*th error that occurred during the execution of a statement run with the NO-ERROR option.

Return type: INTEGER

Applies to: ERROR-STATUS system handle

Syntax

```
GET-NUMBER ( n )
```

n

An integer expression that specifies the error whose number you want to retrieve.

GET-PARENT( ) method

Retrieve the parent node of the node. The first parameter must be a valid X-noderef handle and will refer to the parent XML node if the node has a parent. If the node is the top "root" element in the document, this will return the Unknown value (?).

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
GET-PARENT( x-node-handle )
```

x-node-handle

A valid X-noderef handle to use for the parent XML node.

The following example returns a handle to the parent XML node in hNoderefParent unless the hNoderef is the top "root" element in the hDoc. In that case, it returns the Unknown value (?).

```
hNoderef:GET-PARENT(hNoderefParent)
```

GET-PREV( ) method

Moves a query object's result list pointer back one row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
GET-NEXT ( NO-LOCK | SHARE-LOCK [ , NO-WAIT ]
  | EXCLUSIVE-LOCK [ , NO-WAIT ]  )
```

NO-LOCK

Specifies that no lock is applied to the record. This applies to all buffers in a join.

SHARE-LOCK

Specifies that the record is share locked. This applies to all buffers in a join.

EXCLUSIVE-LOCK

Specifies that the record is exclusively locked. This applies to all buffers in a join.

NO-WAIT

Specifies that the method returns immediately if the record cannot be accessed because it is locked by another user. If you do not use the NO-WAIT option, the method waits until the record can be accessed. This applies to all buffers in a join. If you specify NO-WAIT and the record is locked by another user, the record is returned to you with NO-LOCK and the LOCKED function returns TRUE for the record.

GET-PROPERTY( ) method

Gets the value of the specified application-defined property associated with the Client-principal object. The Client-principal object may be sealed or unsealed. If the specified property is not a valid property for the Client-principal object, or the property does not have a value, this method returns the Unknown value (?).

Return type: CHARACTER

Applies to: Client-principal object handle

Syntax

```
GET-PROPERTY( property-name )
```

property-name

A character string that specifies the name of an application-defined property associated with the Client-principal object. You must enclose this character string in quotes.

You can also use the LIST-PROPERTY-NAMES( ) method to retrieve a list of all application-defined properties associated with the Client-principal object.

Example The following code fragment illustrates how to use the GET-PROPERTY( ) method:

```
DEF VAR hCP as HANDLE.
DEF VAR vVal as CHAR.
.
.
.
CREATE CLIENT-PRINCIPAL hCp.
.
.
.
vVal = hCP:GET-PROPERTY("eye-color").
DISPLAY "Eye color: " vVal.
```

See also LIST-PROPERTY-NAMES( ) method, SET-PROPERTY( ) method

GET-PRINTERS( ) method (Windows only)

Returns a comma-separated list of printers defined in the Windows Registry.

Return type: CHARACTER

Applies to: SESSION system handle

Syntax

```
SESSION:GET-PRINTERS( )
```

If there are no printers defined in the Windows Registry, this method returns the null string (""). Network printers appear in Universal Naming Convention format.

GET-QNAME-BY-INDEX( ) method

Gets the XML qualified name of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-QNAME-BY-INDEX ( index )
```

index

An INTEGER expression indicating the 1-based index of the attribute.

Looks up an attribute's XML qualified name by index. Returns Unknown value (?) if the index is less than 1 or greater than the value of the SAX-attributes object's NUM-ITEMS attribute.

GET-RED-VALUE( ) method (Graphical interfaces only)

Returns the red component of an entry in the color table.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-RED-VALUE ( index )
```

index

An integer expression that specifies an entry in the color table.

GET-RELATION( ) method

Gets the handle of the specified data-relation object.

Return type: HANDLE

Applies to: ProDataSet object handle

Syntax

```
GET-RELATION ( index | relation-name )
```

index

An integer expression indicating the 1-based index of the data-relation object.

relation-name

A character expression that evaluates to the name of the data-relation object.

GET-REPOSITIONED-ROW( ) method

Returns the row index of the browse viewport where the REPOSITION TO ROWID (or RECID) statement displays a repositioned record.

Return type: INTEGER

Applies to: BROWSE widget

Syntax

```
GET-REPOSITIONED-ROW ( )
```

By default, this is the top row in the browse viewport (index 1). Note that this method is only useful in conjunction with the REPOSITION statement. See the SET-REPOSITIONED-ROW( ) method reference entry for more information.

GET-RGB-VALUE( ) method (Graphical interfaces only)

Returns an integer that represents a combination of the red, green, and blue value of an entry in the color table.

Return type: INTEGER

Applies to: COLOR-TABLE system handle

Syntax

```
GET-RGB-VALUE ( index )
```

index

An integer expression that specifies an entry in the color table.

Note: This method is useful primarily when using colors with ActiveX controls.

GET-SELECTED-WIDGET( ) method

Returns the handle of the selected widget in a dialog box, frame, or window.

Return type: WIDGET-HANDLE

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

Syntax

```
GET-SELECTED-WIDGET ( n )
```

n

An integer expression that specifies an index to a selected widget in a frame, dialog box, or window.

You can use the NUM-SELECTED-WIDGETS attribute to determine the total number of selected widgets within the frame or window. The order of the selected widgets is unpredictable.

GET-SERIALIZED( ) method

Returns a LONGCHAR that contains the serialized form of the XML underlying the SOAP header entry or SOAP fault entry.

Return type: LONGCHAR

Applies to: SOAP-fault-detail object handle, SOAP-header-entryref object handle

Syntax

```
GET-SERIALIZED ( )
```

Contents of the returned LONGCHAR are equivalent to taking the X-noderef returned by the GET-NODE( ) method and serializing the data. The caller of this method is responsible for managing the lifetime of the LONGCHAR. Changes made to the contents of the LONGCHAR will NOT be reflected in the original header entry unless it is passed as the parameter in the SET-SERIALIZED( ) method.

Returns the Unknown value (?) if the SOAP-header-entryref object handle has been initialized but does not refer to a header entry (for example, immediately after the CREATE SOAP-HEADER-ENTRYREF statement).

GET-SIGNATURE( ) method

Returns the *signature* of the internal procedure or user-defined function whose name you supply.

Specifically:

- If you provide the name of an internal procedure, GET-SIGNATURE returns the type and mode of each parameter.
- If you provide the name of a user-defined function, GET-SIGNATURE returns the return type, and the type and mode of each parameter.
- If you provide the nil procedure name (""), GET-SIGNATURE returns the signature of the procedure whose handle you supply.
- If you provide a name that does not match any of the internal procedures or user-defined functions in the procedure, GET-SIGNATURE returns the empty string ("").
- If you provide a remote (proxy) procedure handle or the name of a Web service procedure, GET-SIGNATURE returns the empty string ("").
- If you provide the name of a DLL entry point, GET-SIGNATURE returns the Progress equivalent of the C data type of each parameter of the entry point. For more information, see *OpenEdge Development: Programming Interfaces*.

Note: GET-SIGNATURE does not return the signature of any internal procedure defined using the PROCEDURE statement's PRIVATE option. Similarly, GET-SIGNATURE does not return the signature of any user-defined function defined using the FUNCTION statement's PRIVATE option.

Return type: CHARACTER

Applies to: THIS-PROCEDURE system handle (and all procedure handles)

Syntax

```
GET-SIGNATURE ( int-proc-name )
```

int-proc-name

The name of an internal procedure or user-defined function.

GET-SIGNATURE returns a string with the following format:

```
type , return-type ,
  [ mode name p-type [ , mode name p-type ] ... ]
```

type

The type of the internal procedure. Types include:

- **PROCEDURE** — A Progress internal procedure.
- **FUNCTION** — A Progress user-defined function whose definition resides in the procedure.
- **EXTERN** — A Progress user-defined function whose definition resides in another procedure.
- **DLL-ENTRY** — A DLL entry point.
- **MAIN** — The main procedure.

return-type

(User-defined functions only) The Progress data type that a user-defined function returns.

mode name p-type

A parameter description where *mode* is the mode of the parameter, *name* is the name of the parameter, and *p-type* is the type of parameter. The parameter type is either a data type (scalar or array) or, for a buffer parameter, the name of the table associated with the buffer.

The modes are:

- INPUT
- OUTPUT
- INPUT-OUTPUT
- BUFFER
- INPUT TABLE
- OUTPUT TABLE

- INPUT-OUTPUT TABLE

The data types are:

- CHARACTER
- DATE
- DATETIME
- DATETIME-TZ
- DECIMAL
- INTEGER
- LOGICAL
- MEMPTR
- RAW
- RECID
- ROWID
- WIDGET-HANDLE

When you define the parameter type as a determinate array with a constant extent value, the GET-SIGNATURE( ) method returns the constant extent value specified as part of the signature (for example, `EXTENT 100`). When you define the parameter type as a determinate array with a variable extent value, or as an indeterminate array, the GET-SIGNATURE( ) method returns only the extent keyword (that is, `EXTENT`, with no extent value).

GET-SOCKET-OPTION( ) method

Returns a comma separated string containing values appropriate for the specified socket option. Otherwise, it returns the Unknown value (?).

Return type: CHARACTER

Applies to: Socket object handle

Syntax

```
GET-SOCKET-OPTION( name )
```

name

A character expression indicating the name of the socket option to be retrieved. Table 68 describes the options Progress supports.

Table 68: Options for GET-SOCKET-OPTION( ) *(1 of 2)*

Option	Value returned
TCP-NODELAY	An *enable indicator*, which is either TRUE or FALSE.
SO-LINGER	Two comma separated values: • The *onoff indicator*, which is either TRUE or FALSE. • The *linger time*. If the onoff indicator is FALSE, the linger time does not need to be provided.

Table 68: Options for GET-SOCKET-OPTION( ) *(2 of 2)*

Option	Value returned
SO-KEEPALIVE	TRUE if the option is on; FALSE otherwise. The default depends on how the socket object was created: • For socket objects created using CREATE SOCKET, the default is off. • For socket objects created by Progress and passed as a parameter to the event-procedure context, the default is on.
SO-RESUEADDR	TRUE if the option is on; FALSE otherwise. The default depends on the platform.
SO-RCVBUF	An integer that indicates the size of the receive buffer. The default depends on the platform.
SO-SNDBUF	An integer that indicates the size of the send buffer. The default depends on the platform.
SO-RCVTIMEO	The timeout length—that is, the number of seconds you want the socket to wait for expected data before timing out. The default is -1, which tells the socket to wait forever. **Note:** The timeout length is not guaranteed to be precise to the second.

This method returns option-specific data if the retrieval of the option succeeded and the Unknown value (?) otherwise. An error can occur if:

- *name* is not a Progress-supported socket option.
- Getting the socket option fails.

GET-SOURCE-BUFFER( ) method

Gets the handle to the source buffer in the Data-source object at the specified index position.

Return type: HANDLE

Applies to: Data-source object handle

Syntax

```
GET-SOURCE-BUFFER ( [ buffer-index ] )
```

buffer-index

An optional integer expression indicating the 1-based index of the source buffer. The default is 1.

GET-TAB-ITEM( ) method

Returns the handle of a widget at a specified tab position in a field group.

Return type: WIDGET-HANDLE

Applies to: FIELD-GROUP widget

Syntax

```
GET-TAB-ITEM ( n )
```

n

An integer expression that specifies a tab position within a field group.

You can use the MOVE-AFTER-TAB-ITEM( ) and MOVE-BEFORE-TAB-ITEM( ) methods to change the tab position of fields at the field level, and the FIRST-TAB-ITEM and LAST-TAB-ITEM attributes to change the tab positions at the field group level.

If the widget returned is a frame, the specified tab position includes the tab positions of all tab-order widgets contained by the frame. For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry and the chapter on frames in *OpenEdge Development: Progress 4GL Handbook*.

GET-TEXT-HEIGHT-CHARS( ) method (Graphical interfaces only)

Returns the height, in character units, of the specified font. If no font is specified, the method returns the height of the default font.

Return type: DECIMAL

Applies to: FONT-TABLE system handle

Syntax

```
GET-TEXT-HEIGHT-CHARS ( [ font ] )
```

font

An integer expression that specifies an entry within the font.

If you pass the Unknown value (?) to this method, Progress uses the system default font. When a field-level widget inherits its font from the parent frame, Progress returns the Unknown value (?) for the font and you must use the font of the parent frame.

GET-TEXT-HEIGHT-PIXELS( ) method (Graphical interfaces only)

Returns the height, in pixels, of the specified font. If no font is specified, the method returns the height of the default font.

Return type: INTEGER

Applies to: FONT-TABLE system handle

Syntax

```
GET-TEXT-HEIGHT-PIXELS ( [ font ] )
```

font

An integer expression that specifies an entry within the font table.

If you pass the Unknown value (?) to this method, Progress uses the system default font. When a field-level widget inherits its font from the parent frame, Progress returns the Unknown value (?) for the font and you must use the font of the parent frame.

GET-TEXT-WIDTH-CHARS( ) method (Graphical interfaces only)

Returns the width, in character units, of the string using the specified font. If no font is specified, the method calculates the width of the string using the default font.

Return type: DECIMAL

Applies to: FONT-TABLE system handle

Syntax

```
GET-TEXT-WIDTH-CHARS ( string [ , font ] )
```

string

A character-string expression whose width you want to determine.

font

An integer expression that specifies an entry within the font table.

If you pass the Unknown value (?) to this method, Progress uses the system default font. When a field-level widget inherits its font from the parent frame, Progress returns the Unknown value (?) for the font and you must use the font of the parent frame.

GET-TEXT-WIDTH-PIXELS( ) method (Graphical interfaces only)

Returns the width, in pixels, of the string using the specified font. If no font is specified, the method calculates the width of the string using the default font.

Return type: INTEGER

Applies to: FONT-TABLE system handle

Syntax

```
GET-TEXT-WIDTH-PIXELS ( string [ , font ] )
```

string

A character-string expression whose width you want to determine.

font

An integer expression that specifies an entry within the font table.

If you pass the Unknown value (?) to this method, Progress uses the system default font. When a field-level widget inherits its font from the parent frame, Progress returns the Unknown value (?) for the font and you must use the font of the parent frame.

GET-TOP-BUFFER( ) method

Gets the top-level buffer in a ProDataSet object at the specified index position.

Note: A top-level buffer is a ProDataSet object buffer that is not a child in any active data relation. There may be one or more top-level buffers in a ProDataSet object.

Return type: HANDLE

Applies to: ProDataSet object handle

Syntax

```
GET-TOP-BUFFER ( index )
```

index

An integer expression indicating the 1-based index of the top-level buffer.

GET-TYPE-BY-INDEX( ) method

Gets the type of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-TYPE-BY-INDEX ( index )
```

index

An INTEGER expression indicating the 1-based index of the attribute.

The attribute type is one of the following strings: “CDATA,” “ID,” “IDREF,” “IDREFS,” “NMTOKEN,” “NMTOKENS,” “ENTITY,” “ENTITIES,” or “NOTATION.” These are always uppercase.

Returns the Unknown value (?) if the index is less than 1 or greater than the value of SAX-attributes object’s NUM-ITEMS attribute.

GET-TYPE-BY-NAMESPACE-NAME( ) method

Gets the type of the attribute with the given namespace name.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-TYPE-BY-NAMESPACE-NAME ( uri, localname )
```

uri

The namespace URI (that is, the URI to which the attribute’s prefix refers), or, if the name has no namespace URI, an empty string.

localname

The local (unqualified) name of the attribute.

The attribute type is one of the following strings: “CDATA,” “ID,” “IDREF,” “IDREFS,” “NMTOKEN,” “NMTOKENS,” “ENTITY,” “ENTITIES,” or “NOTATION.” These are always uppercase.

Returns the Unknown value (?) if no attribute’s XML qualified name matches the combination of *uri* and *localname* or if namespace processing is disabled.

GET-TYPE-BY-QNAME( ) method

Gets the type of the attribute with the given XML qualified name.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-TYPE-BY-QNAME ( qname )
```

qname

A CHARACTER expression indicating the XML qualified name of the attribute of interest.

The attribute type is one of the following strings: “CDATA,” “ID,” “IDREF,” “IDREFS,” “NMTOKEN,” “NMTOKENS,” “ENTITY,” “ENTITIES,” or “NOTATION.” These are always uppercase.

Returns the Unknown value (?) if no attribute’s XML qualified name matches *qname*.

GET-URI-BY-INDEX( ) method

Gets the namespace URI of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-URI-BY-INDEX ( index )
```

index

An INTEGER expression indicating the 1-based index of the attribute.

Returns the Unknown value (?) if *index* is less than 1 or greater than the value of the SAX-attributes object's NUM-ITEMS attribute.

GET-VALUE-BY-INDEX( ) method

Gets the value of the attribute at the given 1-based index.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-VALUE-BY-INDEX ( index )
```

index

An INTEGER expression indicating the 1-based index of the attribute.

Returns the Unknown value (?) if *index* is less than 1 or greater than the value of the SAX-attributes object's NUM-ITEMS attribute.

GET-VALUE-BY-NAMESPACE-NAME( ) method

Gets the value of the attribute with the given namespace name.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-VALUE-BY-NAMESPACE-NAME ( uri, localname )
```

uri

The namespace URI (that is, the URI to which the attribute's prefix refers), or, if the name has no namespace URI, an empty string.

localname

The local (unqualified) name of the attribute.

Returns the Unknown value (?) if no attribute has a namespace name that matches the *uri* and *localname*, or if namespace processing is disabled.

GET-VALUE-BY-QNAME( ) method

Gets the value of the attribute with the given XML qualified name.

Return type: CHARACTER

Applies to: SAX-attributes object handle

Syntax

```
GET-VALUE-BY-QNAME ( qname )
```

qname

A CHARACTER expression indicating the XML qualified name of the attribute.

Returns the Unknown value (?) if no attribute has a XML qualified name that matches *qname*.

GET-WAIT-STATE( ) method

Returns a string indicating the current wait-state.

Return type: CHARACTER

Applies to: SESSION system handle

Syntax

```
GET-WAIT-STATE ( )
```

If the SET-WAIT-STATE( ) method was called with “GENERAL” or “COMPILER”, GET-WAIT-STATE( ) returns that string. If the SET-WAIT-STATE( ) method was called with an arbitrary pointer name, GET-WAIT-STATE( ) returns “CUSTOM”. If there is no current wait-state, GET-WAIT-STATE( ) returns an empty string ("").

GRAPHIC-EDGE attribute (Character interfaces only)

Indicates whether to draw a rectangle with graphic characters.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: RECTANGLE widget

When the GRAPHIC-EDGE attribute is TRUE, the rectangle is drawn with graphic characters, and the EDGE-CHARS and EDGE-PIXELS attributes are ignored.

GRID-FACTOR-HORIZONTAL attribute (Graphical interfaces only)

The spacing, in horizontal grid units, between the horizontal grid lines of the frame.

Data type: INTEGER

Access: Readable/Writeable

Applies to: FRAME widget

If the value is 1, each horizontal grid unit is intersected by a line of vertical grid points. If the value is greater than 1, every *n*th horizontal grid unit is intersected by a line of vertical grid points, where *n* is the value of GRID-FACTOR-HORIZONTAL. The default value is 6.

The width of a horizontal grid unit is defined by the GRID-UNIT-WIDTH-CHARS or GRID-UNIT-WIDTH-PIXELS attribute.

Note: Setting this attribute to 1 has the same effect as setting the GRID-FACTOR-VERTICAL attribute to 1, because either setting makes all grid points visible in the frame.

GRID-FACTOR-VERTICAL attribute (Graphical interfaces only)

The spacing, in vertical grid units, between the vertical grid lines of the frame.

Data type: INTEGER

Access: Readable/Writeable

Applies to: FRAME widget

If the value is 1, each vertical grid unit is intersected by a line of horizontal grid points. If the value is greater than 1, every *n*th vertical grid unit is intersected by a horizontal line of grid points, where *n* is the value of GRID-FACTOR-VERTICAL. The default value is 6.

The height of a vertical grid unit is defined by the GRID-UNIT-HEIGHT-CHARS or GRID-UNIT-HEIGHT-PIXELS attribute.

Note: Setting this attribute to 1 has the same effect as setting the GRID-FACTOR-HORIZONTAL attribute to 1, because either setting makes all grid points visible in the frame.

GRID-SNAP attribute (Graphical interfaces only)

Indicates whether widgets should snap to the grid when they are moved or resized.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

If the GRID-SNAP attribute is TRUE, when widgets are moved or resized they align with (snap to) the closest grid points in the frame. This alignment occurs whether or not the grid points are visible (determined by the GRID-VISIBLE attribute).

The distance between grid points (vertical and horizontal grid units) is defined by the GRID-UNIT-HEIGHT-CHARS, GRID-UNIT-HEIGHT-PIXELS, GRID-UNIT-WIDTH-CHARS, and GRID-UNIT-WIDTH-PIXELS attributes.

GRID-UNIT-HEIGHT-CHARS attribute (Graphical interfaces only)

The height, in character units, of a vertical grid unit on the frame.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: FRAME widget

This attribute specifies the distance between vertical grid points in the frame. When a widget is moved or resized, it snaps to these grid points within the frame when the GRID-SNAP attribute is set to TRUE. The default value depends on the display resolution and the size of the default system font.

GRID-UNIT-HEIGHT-PIXELS attribute (Graphical interfaces only)

The height, in pixels, of a vertical grid unit on the frame.

Data type: INTEGER

Access: Readable/Writeable

Applies to: FRAME widget

This attribute specifies the distance between vertical grid points in the frame. When a widget is moved or resized, it snaps to these grid points within the frame when the GRID-SNAP attribute is set to TRUE. The default value is 6.

GRID-UNIT-WIDTH-CHARS attribute (Graphical interfaces only)

The width, in character units, of a horizontal grid unit on the frame.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: FRAME widget

This attribute specifies the distance between horizontal grid points in the frame. When a widget is moved or resized, it snaps to these grid points within the frame when the GRID-SNAP attribute is set to TRUE. The default value depends on the display resolution and the size of the default system font.

GRID-UNIT-WIDTH-PIXELS attribute (Graphical interfaces only)

The width, in pixels, of a horizontal grid unit on the frame.

Data type: INTEGER

Access: Readable/Writeable

Applies to: FRAME widget

This attribute specifies the distance between horizontal grid points in the frame. When a widget is moved or resized, it snaps to these grid points within the frame when the GRID-SNAP attribute is set to TRUE. The default value is 6.

GRID-VISIBLE attribute (Graphical interfaces only)

Indicates whether the grid of a frame is visible.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

When visible, the grid is a set of points laid out in vertical and horizontal lines. The distance between grid points (vertical and horizontal grid units), whether visible or invisible, is defined by using the GRID-UNIT-HEIGHT-CHARS, GRID-UNIT-HEIGHT-PIXELS, GRID-UNIT-WIDTH-CHARS, and GRID-UNIT-WIDTH-PIXELS attributes. What grid points are visible is determined by the GRID-FACTOR-VERTICAL and GRID-FACTOR-HORIZONTAL attributes, which define the spacing between the visible vertical and horizontal lines of grid points.

When used with scrollable frames, some grid points might not be visible.

HANDLE attribute

A handle to the object.

Data type: HANDLE

Access: Readable

Applies to: BROWSE widget (browse and cell), Buffer object handle, Buffer-field object handle, BUTTON widget, COMBO-BOX widget, Data-relation object handle, ProDataSet object handle, Data-source object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, Query object handle, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, Server socket object handle, Socket object handle, SUB-MENU widget, Temp-table object handle, TEXT widget, TOGGLE-BOX widget, WEB-CONTEXT system handle, WINDOW widget

You can store this value in a WIDGET-HANDLE variable. You can also use it to associate one widget with another widget or with a system handle. For example, you can assign the HANDLE value of the menu bar to the MENU-BAR attribute of a window, or you can make the window the current window by assigning its HANDLE value to the CURRENT-WINDOW handle.

For query objects, the HANDLE attribute lets you acquire a query object for a static query, as the following fragment demonstrates:

```
my-query-handle = QUERY q:HANDLE.
```

The following code fragment uses the HANDLE attribute of a buffer-field to retrieve the buffer-field's handle:

```
my-buffer-field = city:HANDLE IN BUFFER customer.
```

The preceding code fragment requires that you know the name of the field (in this case, "city") at compile time. The following code fragment, which performs the same task, does not require this:

```
my-buffer = BUFFER customer:HANDLE
my-buffer-field = my-buffer:buffer-field("city").
```

For more information on query, buffer, and buffer-field objects, see *OpenEdge Development: Progress 4GL Handbook*.

HANDLER attribute

A handle to the procedure containing the SAX callbacks.

Data type: HANDLE

Access: Readable/Writable

Applies to: SAX-reader object handle

The default is a handle to the procedure that contains the SAX-PARSE( ) method, or the SAX-PARSE-FIRST( ) and SAX-PARSE-NEXT( ) methods.

When SAX-PARSE( ), SAX-PARSE-FIRST( ), or SAX-PARSE-NEXT( ) executes, the SAX-reader object looks for callbacks only in the procedure whose handle is stored in HANDLER. HANDLER must be a valid procedure handle and cannot be a proxy.

Note: It is permissible for both the driver procedure (the procedure that contains the SAX-PARSE( ) method, or the SAX-PARSE-FIRST( ) and SAX-PARSE-NEXT( ) methods) and handler procedure of a SAX application to reside on a remote AppServer. If this occurs, callbacks are invoked local to the AppServer.

Callbacks can reside within a special handler procedure file that is run persistently or within the driver procedure.

Within a procedure file, to get a handle to the procedure file, use the THIS-PROCEDURE handle. The following fragment assigns HANDLER a handle to the current procedure:

```
hSaxReader:HANDLER = THIS-PROCEDURE.
```

HAS-LOBS attribute

Returns TRUE if the Buffer object has BLOB or CLOB fields defined in it. Otherwise, it returns FALSE.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle

HAS-RECORDS attribute

This attribute returns TRUE when the corresponding temp-table has records. It returns FALSE when the temp-table does not have any records, or the temp-table is in an UNPREPARED state.

Data type: LOGICAL

Access: Readable

Applies to: Temp-table object handle

Height property (Windows only; Graphical interfaces only)

The height of the control-frame and control-frame COM object, in pixels.

Return type: INTEGER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the HEIGHT-CHARS attribute and HEIGHT-PIXELS attribute of the corresponding control-frame widget to an equivalent value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.

HEIGHT-CHARS attribute (Graphical interfaces only)

The height, in character units, of the widget. The HEIGHT-CHARS attribute of the SESSION handle contains the height of the display.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SESSION system handle, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For combo boxes, field groups, and the SESSION handle, this attribute is read-only.

For control-frames, the HEIGHT-CHARS attribute maps to the Height property of the control-frame COM object (ActiveX control container).

For browses, the HEIGHT-CHARS attribute sets the decimal height, in characters, of the browse without changing the height of the browse's rows. If you change the value of a browse's HEIGHT-CHARS or HEIGHT-PIXELS attribute, the number of rows that appear in the viewport might change.

HEIGHT-PIXELS attribute (Graphical interfaces only)

The height, in pixels, of the widget. The HEIGHT-PIXEL attribute of the SESSION handle contains the height of the display.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SESSION system handle, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For combo boxes, field groups, and the SESSION handle, this attribute is read-only.

For control-frames, the HEIGHT-PIXELS attribute maps to the Height property of the control-frame COM object (ActiveX control container).

For browses, the HEIGHT-PIXELS attribute sets the decimal height, in pixels, of the browse without changing the height of the browse's rows. If you change the value of a browse's HEIGHT-CHARS or HEIGHT-PIXELS attribute, the number of rows that appear in the viewport might change.

HELP attribute

The help text for a field.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), Buffer-field object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

For this attribute to have effect, the window that contains the specified widget must have its STATUS-AREA attribute set to TRUE. The text stored in the HELP attribute displays in the status area of the containing window when the widget has input focus. The HELP attribute text overrides any status-area text issued by the STATUS statement.

HIDDEN attribute

Indicates whether to “hide” a widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

Setting the HIDDEN attribute to TRUE prevents the widget from being displayed implicitly. For a field-level widget, child frame, or child window, this means that the widget is not automatically made visible when the containing frame or parent window becomes visible. The widget does not appear unless one of the following occurs:

- It is forced to receive user input (for example, using a SET or PAUSE statement).
- It is explicitly displayed using a VIEW statement or by setting its VISIBLE attribute to TRUE.

Any action that explicitly displays the widget also resets the HIDDEN attribute to FALSE. If the widget is already visible, setting its HIDDEN attribute to TRUE makes that widget and any widgets it parents (and their descendants) invisible (VISIBLE is set to FALSE). The default value of the HIDDEN attribute is FALSE for all widgets.

In windows, setting the HIDDEN attribute to TRUE prevents implicit display of the hidden window when you:

- Invoke DISPLAY, ENABLE, and VIEW statements for frames of the window.
- View an ancestor or descendant window of the hidden window.

This limits flashing side effects caused during set up of the application user interface. In windows, this attribute is not supported in character mode.

For frames and dialog boxes, setting the HIDDEN attribute to TRUE prevents implicit display of the frame or dialog box when you invoke DISPLAY or ENABLE statements for the widget or its descendant frames. This allows the frame or dialog box to remain invisible during actions that set it up. The HIDDEN attribute has no effect on DISPLAY statements directed to a file, pipe, or printer.

Note: Setting a frame or field-level widget's VISIBLE attribute to TRUE also displays any parent or ancestor frames, even if their HIDDEN attributes are set to TRUE (resetting the HIDDEN attributes, if necessary). However, setting a window's VISIBLE attribute to TRUE only displays the window if there are no ancestor windows with their HIDDEN attribute set to TRUE. In any case, the window's own HIDDEN attribute is set to FALSE.

For field-level widgets and frames parented by other frames, setting the HIDDEN attribute to TRUE prevents implicit display of the field-level widget or child frame when its containing frame or dialog box is displayed. If the frame or dialog box containing the widget is visible, setting HIDDEN to FALSE for the widget makes the widget visible (the VISIBLE attribute is set to TRUE). If the containing frame or dialog box is not visible, setting HIDDEN to FALSE has no effect on the VISIBLE attribute of the widget.

Note: The HIDE statement sets the VISIBLE attribute for the widget to FALSE. It only sets the HIDDEN attribute to TRUE if you hide a field-level widget or child frame whose containing frame is still visible.

HonorProKeys property (Windows only; Graphical interfaces only)

Determines who processes the **GO**, **ENDKEY**, **HELP**, and **TAB** keys: Progress, or the ActiveX control to which the property applies.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Any ActiveX control

If the property is TRUE, which is the default, Progress intercepts these keys and processes them as normal Progress key events. If the property is FALSE, the keystrokes are sent to the ActiveX control for processing.

Note: This property resembles the HonorReturnKey property, which governs processing of the **RETURN** key, but whereas the default setting for HonorProKeys is TRUE (Progress gets the event), the default setting for HonorReturnKey is FALSE (the ActiveX control gets the event).

HonorReturnKey property (Windows only; Graphical interfaces only)

Determines who processes the **RETURN** key: Progress, or the ActiveX control to which the property applies.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Any ActiveX control

If the property is TRUE, Progress intercepts the key and processes it as a normal Progress **RETURN** key event. If the property is FALSE, which is the default, the keystroke is sent to the ActiveX control for processing.

Notes

- If a frame has a default button and an ActiveX control, and you want the **RETURN** key to activate the default button regardless of who has focus, you must set the HonorReturnKey property of the ActiveX control to TRUE. Similarly, if a frame has a default button and several ActiveX controls, and you want the **RETURN** key to activate the default button regardless of who has focus, you must set the HonorReturnKey property of all the ActiveX controls in the frame to TRUE.
- This property resembles the HonorProKeys property, which governs processing of several other keys, but whereas the default setting for HonorReturnKey is FALSE (the ActiveX control gets the event), the default setting for HonorProKeys is TRUE (Progress gets the event).

HORIZONTAL attribute

The orientation of a slider, or of radio buttons in a radio set.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: RADIO-SET widget, SLIDER widget

If HORIZONTAL is TRUE, the orientation is horizontal; if it is FALSE, the orientation is VERTICAL. By default, the orientation of sliders is horizontal and the orientation of radio sets is vertical.

You can set this attribute only before the widget is realized.

HTML-CHARSET attribute (WebSpeed only)

The Progress version (as opposed to the MIME version) of the code page name of a Web request. Set by Progress when a WebSpeed application incorporates dynamic code page support. The default is blank.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

Once Progress sets HTML-CHARSET, each time a Web browser sends a Web request to the application:

1. When the request is received by the WebSpeed Agent, Progress converts it from the HTML-CHARSET code page to the Agent's CPINTERNAL code page.

2. When the Agent responds to the request with a Web page, Progress converts the Web page from the Agent's CPINTERNAL code page to the HTML-CHARSET code page.

Caution: If the application modifies the contents of HTML-CHARSET, dynamic code-page support might fail.

For more information on dynamic code-page support, see *OpenEdge Application Server: Developing WebSpeed Applications*.

HTML-END-OF-LINE attribute

Defaults to the newline character (ASCII 10; '~n'; '\n'). A null string value causes a NEWLINE character (not a null string) to be output. You might want to set this to "~n" (the NEWLINE character) or to the null string (to force the NEWLINE character). Depending on the other attribute values, using the NEWLINE rather than the
 tag can result in more readable output when viewing document source in a browser.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-END-OF-PAGE attribute

Between stream pages, defaults to "<HR>". Output between stream pages to visually break up the sectioning caused by the PAGED or PAGE-SIZE options of the OUTPUT TO "WEB" statement. Does not affect the line count of any stream page.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-FRAME-BEGIN attribute

Before a SpeedScript frame, defaults to "<PRE>". Generally, if you change this value you must change the value of HTML-FRAME-END. Output only before the data row(s) for the current iteration of a DOWN frame, not to column headers (see also HTML-HEADER-BEGIN and HTML-HEADER-END). Applies to any side-labels displayed in the frame, whether or not the frame is a DOWN frame.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-FRAME-END attribute

After a SpeedScript frame, defaults to "</PRE>". Generally, if you change this value you must change the value of HTML-FRAME-BEGIN. Output at the end of the data row(s) for the current iteration of a DOWN frame.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-HEADER-BEGIN attribute

Before the column headers of a SpeedScript frame, defaults to "<PRE>". Generally, if you change this value you must change the value of HTML-HEADER-END. Output at the beginning of the column header section of a DOWN frame.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-HEADER-END attribute

After the column headers of a SpeedScript frame, defaults to "</PRE>". Generally, if you change this value you must change the value of HTML-HEADER-BEGIN. Output at the end of the column header section of a DOWN frame.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-TITLE-BEGIN attribute

Before a SpeedScript frame title, Defaults to the null string (""), no text. Generally, if you change this value you must change the value of HTML-TITLE-END. Output before the frame's TITLE value. Setting to a color or bold tag might improve readability.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HTML-TITLE-END attribute

After a SpeedScript frame title, defaults to the null string (""), no text. Generally, if you change this value you must change the value of HTML-TITLE-BEGIN. Output after the frame's TITLE value.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

HWND attribute (Windows only; Graphical interfaces only)

An integer value for a Windows handle to the window that contains the widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is supported for dynamic link library (DLL) access only in Windows. Some DLL routines require that you pass this value.

For Progress window widgets, the Windows window that contains the widget is actually the parent of the Windows widget referenced by HWND. Thus, to obtain the handle of the Windows window that contains the Progress window, you must pass the value of HWND to the `GetParent( )` function (in the `user32.dll`). Pass the result of `GetParent( )` to the DLL routine that requires it.

ICFPARAMETER attribute

A character string that supplies Internet Component Framework (ICF) procedures (in Progress Dynamics®) with ICF-related data.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

Use the ICF Parameter (-icfparam) parameter to specify a character string, at the start of an OpenEdge session, that can be accessed from 4GL procedures within the ICF framework.

Note: The ICF Parameter (-icfparam) parameter is reserved for use by the ICF and procedures that have been integrated with the ICF. Using this parameter for any purpose other than operating within the ICF framework will interfere with your ability to integrate your application with that framework at a later time.

ICON attribute

Returns the name of the icon loaded by LOAD-ICON( ).

Data type: CHARACTER

Access: Readable

Applies to: WINDOW widget

IGNORE-CURRENT-MODIFIED attribute

This attribute is supported only for backward compatibility. Use the PREFER-DATASET attribute instead.

IMAGE attribute

Returns the name of the image loaded by LOAD-IMAGE( ).

Data type: CHARACTER

Access: Readable

Applies to: BUTTON widget, IMAGE widget

IMAGE-DOWN attribute

Returns the name of the image loaded by LOAD-IMAGE-DOWN( ).

Data type: CHARACTER

Access: Readable

Applies to: BUTTON widget

IMAGE-INSENSITIVE attribute

Returns the name of the image loaded by LOAD-IMAGE-INSENSITIVE( ).

Data type: CHARACTER

Access: Readable

Applies to: BUTTON widget

IMAGE-UP attribute

Returns the name of the image loaded by LOAD-IMAGE( ) or LOAD-IMAGE-UP( ).

Data type: CHARACTER

Access: Readable

Applies to: BUTTON widget

IMMEDIATE-DISPLAY attribute (Graphical interfaces only)

The frequency of screen updates for the current session.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, Progress updates the display for every I/O operation, including DISPLAY statements. If FALSE, Progress does not update the display until a statement blocks for input, such as an UPDATE statement. FALSE is the default setting. A TRUE setting provides more accurate screen displays during long display loops at the price of slower performance.

In Windows, this attribute has a similar effect on interactive I/O to setting the MULTITASKING-INTERVAL attribute to a low non-zero value, providing a more frequent display refresh. This attribute also provides the same functionality as the ImmediateDisplay parameter in the current environment (which might be the Registry (Windows only) or an initialization file). For more information on environments, see the chapter on user interface environments in *OpenEdge Deployment: Managing 4GL Applications*.

IMPORT-NODE( ) method

Import a copy of a node from another document into this document. The first parameter must be a valid X-noderef handle and will refer to the newly copied XML node if the method succeeds. The new node is associated with this document, but must be appended or inserted with APPEND-CHILD( ) or INSERT-BEFORE( ) to become part of the structure.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
IMPORT-NODE( x-node , x-source-node , deep )
```

x-node

A valid X-noderef handle to use for the new XML node.

x-source-node

A valid X-noderef handle that represents the node to import from.

deep

A logical that if TRUE specifies that the whole sub-tree is to be copied. The default value is FALSE.

If hDoc is an existing and loaded X-document and hDocCopy is existing but empty and hRoot and hRootCopy are X-noderefs, you can copy hDoc to hDocCopy as follows:

```
/* Associates hRoot with the root node of the hDoc document. */
hDoc:GET-DOCUMENT-ELEMENT(hRoot).
hDocCopy:IMPORT-NODE(hRootCopy,hRoot,true).
hDocCopy:APPEND-CHILD(hRootCopy).
```

IMPORT-PRINCIPAL( ) method

Imports a sealed Client-principal object, from its raw data form, and assigns it to a Client-principal object handle. The receiving Client-principal object handle assumes the property and attribute settings defined in the source object when it was exported using the EXPORT-PRINCIPAL( ) method.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
IMPORT-PRINCIPAL( expression )
```

expression

A RAW expression containing the Client-principal object data to import. If the specified expression has the Unknown value (?), Progress generates a run-time error.

The receiving Client-principal object must be unsealed. If not, Progress generates a run-time error.

If you set properties on the receiving Client-principal object handle before calling this method, either by having previously imported the object or by setting them individually, the property values are lost.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

You can use the imported Client-principal object to set a user ID using either the SET-CLIENT( ) method or SET-DB-CLIENT function.

Calling this method does not generate an audit event or an audit record.

Example The following code fragment illustrates how to use the IMPORT-PRINCIPAL( ) method:

```
DEF VAR hCP as HANDLE.
DEF VAR raw-cp as RAW.
.
.
.
CREATE CLIENT-PRINCIPAL hCP.
.
.
.
hCP:IMPORT-PRINCIPAL(raw-cp).
```

See also EXPORT-PRINCIPAL( ) method, SET-CLIENT( ) method, SET-DB-CLIENT function

INCREMENT-EXCLUSIVE-ID( ) method

Gets the amount by which to increment the exclusive ID of a Web request for state-aware agents. Do not access this method.

Return type: INTEGER

Applies to: WEB-CONTEXT system handle

INDEX attribute

The subscript value of the array element referenced by the current widget.

Data type: INTEGER

Access: Readable

Applies to: FILL-IN widget, TEXT widget

If the widget references a field or variable with no extents (that is, not as an array element), this attribute returns 0.

INDEX-INFORMATION attribute

A character string consisting of a comma-separated list of the index or indexes the query uses at the level of join specified.

Data type: CHARACTER

Access: Readable

Applies to: Query object handle

Syntax

```
INDEX-INFORMATION ( n )
```

n

An INTEGER expression that evaluates to the level of join for which you want index information.

If the index or indexes do not have bracketing, the first entry in the list is the CHARACTER string “WHOLE-INDEX,” and the second entry in the list is name of the index.

Before you use INDEX-INFORMATION on a dynamic query, you must prepare the query using the QUERY-PREPARE method.

Before you can use the INDEX-INFORMATION attribute on a static query, you must define the query using the DEFINE QUERY statement’s RCODE-INFORMATION option.

The following example prints out the PREPARE-STRING, analyzes the INDEX-INFORMATION, and prints a list of bracketed and whole-index indexes:

r-iinfo.p

```
/* r-iinfo.p */

def query q for customer,order,order-line scrolling.
def var x as handle.
def var i as integer.
def var j as integer.

x = query q:handle.

x:query-prepare("for each customer where cust-num < 3,
  each order of customer, each order-line").
x:query-open.

message "prepare string is" x:prepare-string.

repeat i = 1 to x:num-buffers:
  j = lookup("WHOLE-INDEX",x:index-information(i)).
  if (j > 0)
    then message "inefficient index"
                 entry(j + 1,x:index-information(i)).
    else message "bracketed index use of" x:index-information(i).
end.
```

INDEX-INFORMATION( ) method

Returns index information in a comma-separated list for the i^{th} index in the buffer's table.

The returned comma-separated list consists of the following in the specified order:

- The index name.
- Three integers of value 0 (FALSE) or 1 (TRUE) depending on whether the index is unique, primary or a word index.
- The names of the index fields, each followed by a 0 (ascending) or 1 (descending).

Return type: CHARACTER

Applies to: Buffer object handle

Syntax

```
INDEX-INFORMATION(i)
```

i

The relative number of the buffer table's index for which you want information.

Note When the index argument, *i*, is beyond the number of indices in the table or is otherwise invalid, the Unknown value (?) is returned.

Example The following code fragment requests information about the third index in the customer table:

```
buffCustHdl = BUFFER customer:HANDLE.
IndexVar = buffCustHdl:INDEX-INFORMATION(3).
```

The returned string would look like: "cust-num,1,1,0,cust-num,0" which means that the third index in the customer table is called "cust-num". It is unique and primary, and is not a word index and it consists of one ascending component, cust-num.

INITIAL attribute

The value of the INITIAL schema field (which is always CHARACTER), formatted with the buffer-field's format. If the INITIAL schema field has the Unknown value (?), the value of the INITIAL attribute is the null string ("").

Data type: CHARACTER

Access: Readable

Applies to: Buffer-field object handle

INITIALIZE-DOCUMENT-TYPE( ) method

Creates a new XML document, initializes the document based on the referenced DTD, and creates its root node.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
INITIALIZE-DOCUMENT-TYPE( namespace-uri , root-node-name , public-id ,
system-id )
```

namespace-uri

A character expression representing the namespace Uniform Resource Identifier (URI) you want associated with the root node of the XML document. The *namespace-uri* must be unique and persistent.

root-node-name

A character expression representing the name of the root node as defined in the XML document. If you are using namespaces and you want to associate a prefix with the namespace, you must qualify this node name with the *namespace-uri* and a colon character prefix (for example, *namespace-uri:root-node-name*). You must explicitly set the xmlns attribute on the root node.

public-id

An optional character expression representing the public ID of the DTD. Currently, there is no way to retrieve a DTD based on a public ID.

system-id

A required character expression representing the system ID of the DTD. This contains the path to the DTD which is either a file system path or an HTTP URL. The Progress parser uses this information to retrieve the DTD when parsing the document.

Example

The following example initializes an X-document with a DTD reference and adds the proper namespace declaration, if the namespace URI is not empty:

```
DEFINE INPUT PARAMETER namespaceURI AS CHARACTER NO-UNDO.
DEFINE INPUT PARAMETER rootNodeName AS CHARACTER NO-UNDO.
DEFINE INPUT PARAMETER publicId AS CHARACTER NO-UNDO.
DEFINE INPUT PARAMETER systemId AS CHARACTER NO-UNDO.
DEFINE OUTPUT PARAMETER hDocument AS HANDLE NO-UNDO.
DEFINE VARIABLE hNsDecl AS HANDLE NO-UNDO.
DEFINE VARIABLE hRootNode AS HANDLE NO-UNDO.
DEFINE VARIABLE errStat AS LOGICAL NO-UNDO.
DEFINE VARIABLE found AS INTEGER NO-UNDO.

/* Create X-document and intialize it. */
CREATE X-DOCUMENT hDocument.
errStat = hDocument:INITIALIZE-DOCUMENT-TYPE(namespaceURI, rootNodeName,
publicId, systemId).
IF errStat = NO THEN
DO:
   DELETE OBJECT hDocument.
   LEAVE.
END.

/* If using namespaces, create X-NODEREF for namespace declaration. */
IF LENGTH(namespaceURI) > 0 THEN
DO:
   CREATE X-NODEREF hNsDecl.
   CREATE X-NODEREF hRootNode.

   /* Look for a colonized name in rootNodeName. */
   found = INDEX(rootNodeName, ":").
   IF found > 0 THEN
   DO:
      /* Namespace declarations are special kinds of attributes that */
      /* belong in the http://www.w3.org/2000/xmlns/ namespace.*/
      errStat = hDocument:CREATE-NODE-NAMESPACE(hNsDecl,
      "http://www.w3.org/2000/xmlns/", "xmlns:" +
      SUBSTRING(rootNodeName, 1, found - 1), "attribute").
   END.
   ELSE
```

```
   DO:
      /* Use the default namespace, which does not need a */
      /* namespace declaration prefix, and assign it to the */
      /* http://www.w3.org/2000/xmlns/ namespace.*/
      errStat = hDocument:CREATE-NODE-NAMESPACE(hNsDecl,
      "http://www.w3.org/2000/xmlns/", "xmlns", "attribute").
   END.
   IF errStat = NO THEN
      LEAVE.

   /* Set the value of the namespace attribute to the namespace URI. */
   hNsDecl:NODE-VALUE = namespaceURI.

   /* Retrieve the root node and add the namespace declaration to it. */
   errStat = hDocument:GET-DOCUMENT-ELEMENT(hRootNode).
   IF errStat = NO THEN
      LEAVE.
   errStat = hRootNode:SET-ATTRIBUTE-NODE(hNsDecl).
END.

/* Free up the temporary X-NODEREFS. */
IF VALID-HANDLE(hNsDecl) THEN
   DELETE OBJECT hNsDecl.
IF VALID-HANDLE(hRootNode) THEN
   DELETE OBJECT hRootNode.

/* If an error occurred, free up the X-document. */
IF errStat = NO THEN
   DELETE OBJECT hDocument.
```

INITIATE( ) method

Initializes the Debugger, but does not pass control to the Debugger immediately. To start the Debugger from the procedure in application mode, you must set a breakpoint using the SET-BREAK( )method that the procedure encounters. When the procedure encounters the breakpoint, the Debugger takes control of the procedure at that point.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
INITIATE ( )
```

If the INITIATE( ) method successfully initializes the Debugger or the Debugger is already initialized, the method returns TRUE. Otherwise, it returns FALSE with no effect.

Note: To use this method, the Application Debugger must be installed.

All other Debugger attributes and methods (except the DEBUG( ) method) have no effect unless you first initialize the Debugger with this method or start the Debugger from the OpenEdge ADE. If the Debugger is already initialized and running (for example, by running Progress with the `-debug` startup parameter), this method has no effect.

INNER-CHARS attribute

The number of data columns within a selection list or editor widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: EDITOR widget, SELECTION-LIST widget

This attribute is more portable than the WIDTH-CHARS attribute.

For editor widgets, this attribute can set the word wrap margin for the WORD-WRAP attribute. For more information, see the WORD-WRAP attribute reference entry.

If a selection list is not already realized and you reference its INNER-CHARS attribute, Progress realizes the widget.

INNER-LINES attribute

The number of data lines within a combo-box drop down list, editor widget, or selection list.

Data type: INTEGER

Access: Readable/Writeable

Applies to: COMBO-BOX widget, EDITOR widget, SELECTION-LIST widget

For combo boxes, this attribute has meaning only in Windows.

This attribute is more portable than the HEIGHT-CHARS attribute. For combo boxes, the value must be 3 or greater.

If a selection list or combo box is not already realized and you reference its INNER-LINES attribute, Progress realizes the widget.

INPUT-VALUE attribute

Used for data-representation widgets, such as field-level widgets that represent variables or database fields. The value for the INPUT-VALUE attribute is the unformatted SCREEN-VALUE of a widget.

Data type: Same as the field or variable associated with the widget

Access: Readable

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget, LITERAL widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

For any widget that has a SCREEN-VALUE, INPUT-VALUE returns the unformatted value of the widget's SCREEN-VALUE, in the native data type of the widget.

The INPUT-VALUE attribute has the following syntax:

```
<widget-name>:INPUT-VALUE
```

When you use the INPUT-VALUE attribute, there are no formatting characters and the data type is the data type of the associated field. The relationship between a field's INPUT-VALUE and the value in the record buffer is the same as the relationship between the SCREEN-VALUE and the record buffer. For example, changing the record buffer does not affect the INPUT-VALUE.

When you use the DISPLAY, PUT, or MESSAGE statements for the INPUT-VALUE of a widget that has the DATE data type, you see the formatted value because these statements automatically format DATE values. To display an unformatted date, first assign INPUT-VALUE to an INT variable, and then display the variable.

See also SCREEN-VALUE attribute

INSERT( ) method

Inserts a new item before a specified item in a combo box or selection list. The new item can consist of a label, a list of labels, or a label-value pair.

Return type: LOGICAL

Applies to: COMBO-BOX widget, SELECTION-LIST widget

Syntax

```
INSERT (
  { new-item-list | new-label , new-value }
  ,
  { list-item | list-index }
  )
```

new-item-list

A character-string expression that specifies a single item or a delimiter-separated list of items to add to the widget.

new-label

A character-string expression that specifies the label of a label-value pair to add to the widget.

new-value

The new value assigned when a user selects the label.

Note: If the widget's entries consist of single items, use *new-item-list*. If the widget's entries consist of label-value pairs, use *new-label* and *new-value*.

list-item

A character-string expression that specifies a single value in the widget.

list-index

An INTEGER expression that specifies the ordinal position of an existing entry in the widget. The first item is specified by 0 and the last item is specified by -1.

The delimiter is the value of the DELIMITER attribute, which is a comma by default. If the method is successful, it returns TRUE.

Note: If the widget's entries consist of single items, each call to INSERT can add multiple entries. If the widget's entries consist of label-value pairs, each call to INSERT can add one entry.

INSERT-ATTRIBUTE( ) method

Adds a single attribute to a start tag in an XML document represented by a SAX-writer object.

Return Type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
INSERT-ATTRIBUTE( name, value [ , namespace-URI ] )
```

name

A LONGCHAR expression evaluating to the fully qualified or unqualified name of the attribute.

value

A LONGCHAR expression evaluating to the value of the attribute.

namespace-URI

A LONGCHAR expression evaluating to the URI of the attribute. If the attribute doesn't contain a namespace, then set it to an empty string ("") or the Unknown value (?).

Call this method to add a simple, single attribute to a start tag. You can only call this method immediately after a call to the START-ELEMENT, EMPTY-ELEMENT, INSERT-ATTRIBUTE, or DECLARE-NAMESPACE method. That is, you can only call this method when the WRITE-STATUS is SAX-WRITE-TAG. After calling this method, the status remains SAX-WRITE-TAG.

Regardless of the value of the STRICT attribute, this method fails if you do not call it following one of the valid methods listed above.

If you use *namespace-URI*, then the method resolves the prefix in the following order:

- It attempts to extract the namespace from *name*.
- It attempts to extract the namespace from a previously declared namespace.

If the method call only contains *name* and that value contains a prefix, then the SAX-writer attempts to resolve the prefix to a namespace. If it fails to resolve the namespace and the STRICT attribute is TRUE, then the method fails.

Namespace declarations appear as attributes in an XML document. It is possible to declare a namespace by using INSERT-ATTRIBUTE. To declare a namespace with INSERT-ATTRIBUTE, the special prefix for namespaces, `xmlns`, you must use the prefix with the name of the attribute, and the namespace URI must be the value of the attribute.

Example The following code fragment illustrates how to declare a namespace with the INSERT-ATTRIBUTE method:

```
swh:START-ELEMENT("prefix:name").
swh:INSERT-ATTRIBUTE("xmlns:prefix", "target.url").
swh:END-ELEMENT("prefix:name").

/* Contrast the example above with the standard way to declare a namespace. */
swh:START-ELEMENT("prefix:name").
swh:DECLARE-NAMESPACE("target.url", "prefix").
swh:END-ELEMENT("prefix:name").
```

Namespaces declared using INSERT-ATTRIBUTE in this way are considered namespaces and checked with other namespaces during the write of the XML document. If the `xmlns` prefix does not appear, then attribute is inserted as a regular attribute and it is not handled with the namespace checking during the write.

See also DECLARE-NAMESPACE( ) method

INSERT-BACKTAB( ) method (Character interfaces only)

Moves the cursor backward to the previous four-space tab stop without affecting the text in the widget.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
INSERT-BACKTAB ( )
```

If the operation is successful, the method returns TRUE.

INSERT-BEFORE( ) method

Insert a node as a child of this document before another node (or last if the other node is unknown). This is one way to place the node into the document structure after the node has been created with the CREATE-NODE( ) or CREATE-NODE-NAMESPACE( ) method, cloned with the CLONE-NODE( ) method, or removed with the REMOVE-NODE( ) method. (Similar to the APPEND-CHILD( ) method.)

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
INSERT-BEFORE( x-ref-handle1 , x-ref-handle2 )
```

x-ref-handle1

The handle that represents the node to insert as a child of this document.

x-ref-handle2

A handle that represents the XML node that the node is to be inserted before. If unknown, the node will be appended as the last child.

The following code fragment demonstrates the use of the INSERT-BEFORE( ) method. hNoderefParent ends up with hNoderef and hNoderef2 in that order:

```
. . .
hDoc:CREATE-NODE(hNoderef,bufField:NAME,"ELEMENT").
hNoderefParent:INSERT-BEFORE(hNoderef,hNoderef2).
. . .
```

INSERT-FILE( ) method

Inserts the text of *filename* into the editor widget at the current location of the text cursor.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
INSERT-FILE ( filename )
```

filename

A character-string expression of the full or relative pathname of a file.

If the text insertion is successful, the method returns TRUE. The INSERT-FILE( ) method searches the PROPATH to find the file.

Note: This method replaces each horizontal tab character with eight spaces as it inserts the text into the widget.

INSERT-ROW( ) method

Inserts a blank line in an updateable browse before or after the last selected row. The blank line is a placeholder for a new record to be added through the browse. This method cannot be used with a read-only browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
INSERT-ROW ( BEFORE | AFTER )
```

BEFORE

Adds a new row before the current browse row. This is the default.

AFTER

Adds a new row after the current browse row.

INSERT-STRING( ) method

Inserts a string into the editor widget at the current location of the text cursor.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
INSERT-STRING ( string )
```

string

A character string expression.

If the operation is successful, the method returns TRUE.

INSERT-TAB( ) method (Character interfaces only)

This method works differently depending on the insert mode status. If insert mode is on, it inserts one to four spaces from the current cursor position to the next four-space tab stop. If insert mode is off, it moves the cursor to the next four-space tab stop without inserting any characters.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
INSERT-TAB ( )
```

If the operation is successful, the method returns TRUE.

INSTANTIATING-PROCEDURE attribute

Returns the handle to the procedure in which an object was instantiated.

Data type: HANDLE

Access: Readable

Applies to: ACTIVE-WINDOW system handle, Asynchronous request object handle, BROWSE widget (browse and column), Buffer object handle, Buffer-field object handle, BUTTON widget, CALL object handle, COMBO-BOX widget, CONTROL-FRAME widget, CURRENT-WINDOW system handle, Data-relation object handle, Data-source object handle, DEFAULT-WINDOW system handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, ProDataSet object handle, Query object handle, RADIO-SET widget, RECTANGLE widget, SAX-attributes object handle, SAX-reader object handle, SELECTION-LIST widget, SELF system handle, Server object handle, Server socket object handle, SLIDER widget, Socket object handle, SOURCE-PROCEDURE system handle, SUB-MENU widget, TARGET-PROCEDURE system handle, Temp-table object handle, TEXT widget, THIS-PROCEDURE system handle, TOGGLE-BOX widget, Transaction object handle, WEB-CONTEXT system handle, WINDOW widget, X-document object handle, X-noderef object handle

When applied to a widget or object handle, this attribute always returns a handle. This handle is either valid or invalid based on the state of the instantiating procedure. The instantiating procedure for a dynamically created object might no longer be running. In this case, the handle is invalid.

Note: Since you can never be sure of the state of an instantiating procedure, never use this attribute as a chained attribute.

In general, INSTANTIATING-PROCEDURE returns the procedure handle for the procedure in which the DEFINE or CREATE statement is executed. Following are the exceptions:

- For a procedure handle or an asynchronous request object handle, INSTANTIATING-PROCEDURE returns the procedure handle for the procedure from which the RUN statement was executed.
- For a socket handle created on a socket server after a CONNECT statement, INSTANTIATING-PROCEDURE returns the procedure handle for the procedure in which the socket server connect procedure is defined.
- For a temp-table handle that is passed from a client to an Appserver, INSTANTIATING-PROCEDURE returns the procedure handle for the first procedure run on the AppServer.
- For a temp-table handle that is received from an AppServer by a client, INSTANTIATING-PROCEDURE returns the procedure handle for the procedure that called the AppServer.
- For implicitly created handles, INSTANTIATING-PROCEDURE returns the procedure handle where the implicitly created object is returned (such as the SAX-attributes object in the StartElement callback).
- For the COM-SELF system handle, which is not a widget handle, INSTANTIATING-PROCEDURE returns a run-time error.

Although this attribute applies to all widgets and handles, not all system handles are associated with an instantiating procedure and do not return a procedure handle. For these system handles, this attribute returns the Unknown value (?). The following system handles always return the Unknown value (?):

CLIPBOARD system handle	CODEBASE-LOCATOR system handle
COLOR-TABLE system handle	COMPILER system handle
DEBUGGER system handle	ERROR-STATUS system handle
FILE-INFO system handle	FOCUS system handle
FONT-TABLE system handle	LAST-EVENT system handle
RCODE-INFO system handle	SESSION system handle
WEB-CONTEXT system handle	

INTERNAL-ENTRIES attribute

A comma-separated list containing the names of all internal procedures and user-defined functions defined in the procedure associated with the specified handle. Returns the Unknown value (?) for a Web service procedure or proxy persistent procedure.

Data type: CHARACTER

Access: Readable

Applies to: THIS-PROCEDURE system handle (and all procedure handles)

If you supply a handle to a procedure that defines no internal procedures or user-defined functions, the value of INTERNAL-ENTRIES is the null string ("").

For more information on proxy handles and remote procedures, see *OpenEdge Application Server: Developing AppServer Applications*.

Note: The list provided by INTERNAL-ENTRIES does not contain the name of any internal procedure defined using the PROCEDURE statement's PRIVATE option. Similarly, the list does not contain the name of any user-defined function defined using the FUNCTION statement's PRIVATE option.

INVOKE( ) method

Lets you do the following dynamically:

- Invoke an external procedure, internal procedure, or user-defined function.
- Get or set an attribute.
- Run a method.

Return type: None (Similar to the RUN statement.)

Applies to: CALL object handle

Syntax

```
INVOKE( )
```

To determine what action to take, INVOKE( ) examines the CALL-NAME and CALL-TYPE attributes.

Before you execute INVOKE( ), you must set the CALL-NAME attribute.

Table 69 describes what INVOKE( ) does for each call type.

Table 69: What INVOKE( ) does for each call type

For this call type ...	INVOKE( ) ...
PROCEDURE-CALL-TYPE (the default)	Follows the rules of the RUN statement to determine what to invoke and whether to invoke external or internal procedures.
FUNCTION-CALL-TYPE	Follows the rules of the DYNAMIC-FUNCTION( ) function.
GET-ATTR-CALL-TYPE	Follows the rules of *widget:attribute*. **Note:** The IN-HANDLE attribute must be set before INVOKE( ) is executed.
SET-ATTR-CALL-TYPE	Sets the attribute specified by the CALL-NAME attribute to the first parameter specified by the SET-PARAMETER( ) method. **Note:** The IN-HANDLE attribute must be set before INVOKE( ) is executed.

If the PERSISTENT attribute is TRUE (which is valid only for invoking an external procedure), the procedure runs persistently, and when INVOKE( ) returns, IN-HANDLE contains a handle to the persistent procedure.

Before you dynamically invoke an external procedure that is remote—that is, one that resides on an AppServer—you must set the SERVER attribute to the handle of the AppServer.

When INVOKE( ) starts executing, it examines the NUM-PARAMETERS attribute. If NUM-PARAMETERS is nonzero, INVOKE( ) uses each parameter set by the SET-PARAMETER( ) method, even one set during a previous use of the CALL object.

Note: To clear all parameters, even those set during a previous use of the CALL object, set the NUM-PARAMETERS attribute to zero.

No parameters are evaluated during INVOKE( ) processing. Parameters are evaluated only during SET-PARAMETER( ) processing.

When the invoked routine starts, if any parameter indicated by the NUM-PARAMETERS attribute has not been set, you will get an error message.

For more information on INVOKE( ), see the reference entries for the RUN statement and the DYNAMIC-FUNCTION( ) function.

Note: INVOKE( ) cannot occur within an expression.

IN-HANDLE attribute

A handle to one of the following:

- A persistent procedure you just started up dynamically (by invoking an external procedure with the PERSISTENT attribute set to TRUE).
- A running persistent procedure containing an internal procedure or user-defined function you want to invoke dynamically.
- An object whose attributes you want to get or set dynamically, or whose methods you want to run dynamically.

Data type: HANDLE

Access: Readable/Writable

Applies to: CALL object handle

Syntax

```
IN-HANDLE [ = { handle-expression | char-expression } ]
```

handle-expression

A HANDLE expression. Table 70 explains what the handle can indicate and who sets it.

Table 70: What IN-HANDLE indicates and who sets it

If the handle indicates...	It is set by...
A persistent procedure you just started up dynamically.	The INVOKE( ) method
An already-running persistent procedure containing logic (in the form of internal procedures and user-defined functions) you want to invoke dynamically.	You
An object whose attributes you want to get or set dynamically.	You

char-expression

A CHARACTER expression indicating the name of a system object, such as “SESSION” or “FILE-INFO.”

The default is the Unknown value (?).

When you use IN-HANDLE to call an internal procedure, IN-HANDLE affects INVOKE( ) the same way the IN *proc-handle* phrase affects the RUN statement. Similarly, when you use IN-HANDLE to call a user-defined function, it affects INVOKE( ) the same way the IN *proc-handle* phrase affects the DYNAMIC-FUNCTION function. In both cases, IN-HANDLE specifies the instance of the external procedure that contains the internal procedure or user-defined function.

When IN-HANDLE is used to get or set an attribute or to invoke a method, it represents a handle to the object to which the attribute applies. If the attribute applies to a system object such as the SESSION handle or the FILE-INFO handle, IN-HANDLE can be set to a character string such as “SESSION” or “FILE-INFO” that indicates the name of the system object.

Note: When you create a running persistent procedure by running an external procedure persistently, you can do this statically or dynamically. Similarly, you can run any of the persistent procedure’s internal procedures and user-defined functions statically or dynamically.

For information on dynamically invoking logic that resides on an AppServer, see the reference entry for SERVER attribute.

IS-OPEN attribute

Indicates whether a transaction or query object is open.

Data type: LOGICAL

Access: Readable

Applies to: Query object handle, Transaction object handle

The IS-OPEN attribute is TRUE if the specified database transaction or query object is active.

For transaction handles, this attribute is identical to the TRANSACTION function.

IS-PARAMETER-SET attribute

Indicates whether you have already set a particular parameter.

Data type: LOGICAL

Access: Readable

Applies to: CALL object handle

Syntax

```
IS-PARAMETER-SET(parameter-number)
```

parameter-number

An INTEGER expression indicating the order of the parameter, where 1 represents the first parameter, 2 represents the second parameter, and so on.

If the parameter is set, IS-PARAMETER-SET returns TRUE. Otherwise, it returns FALSE. The default is FALSE.

IS-ROW-SELECTED( ) method

Returns TRUE if a specified row in the browse viewport is currently selected.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
IS-ROW-SELECTED ( n )
```

n

An integer expression that specifies a selected row within the browse viewport.

Progress maintains a numbered list of selected rows, starting at 1.

IS-SELECTED( ) method

Returns TRUE if a specified item in a selection list is currently selected. Otherwise, the method returns FALSE.

Return type: LOGICAL

Applies to: SELECTION-LIST widget

Syntax

```
IS-SELECTED ( list-item | list-index )
```

list-item

A character-string expression that specifies a single value in the selection list.

list-index

An INTEGER expression that specifies the ordinal position (first, second, third, etc.) of an entry in the selection-list.

IS-XML attribute

Returns whether an XML document was posted to the transaction server.

Data type: LOGICAL

Access: Readable

Applies to: WEB-CONTEXT system handle

ITEMS-PER-ROW attribute

How to format multiple items written to the system clipboard using the CLIPBOARD handle.

Data type: INTEGER

Access: Readable/Writeable

Applies to: CLIPBOARD system handle

This attribute has meaning only when the CLIPBOARD:MULTIPLE attribute is TRUE, and has no effect when reading data from the clipboard. For more information, see the reference entry for the CLIPBOARD system handle.

KEEP-CONNECTION-OPEN attribute

Indicates whether WebClient should keep any server connection, that it creates, open after downloading a file (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Readable

Applies to: CODEBASE-LOCATOR system handle

KEEP-FRAME-Z-ORDER attribute

The overlay order of the frames in a window.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

If the KEEP-FRAME-Z-ORDER attribute is set to TRUE, Progress ignores the default frame overlay behavior.

By default, Progress moves the frame that contains the field with focus to the top. When you set the KEEP-FRAME-Z-ORDER attribute to TRUE, you are responsible for maintaining the overlay order of the frames using the MOVE-TO-TOP( ) and MOVE-TO-BOTTOM( ) methods. You should always set this attribute to TRUE when you use the MOVE-TO-TOP( ) and MOVE-TO-BOTTOM( ) methods.

The default value is FALSE.

KEEP-SECURITY-CACHE attribute

Indicates whether WebClient saves the values of the attributes in the security cache between sessions (TRUE) or not (FALSE), as requested by the user. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

The following attributes represent the security cache: APPSERVER-INFO, APPSERVER-PASSWORD, APPSERVER-USERID, URL-PASSWORD, URL-USERID, and KEEP-SECURITY-CACHE. These attributes are readable and writeable.

KEY attribute

Indicates whether the field corresponding to a buffer-field participates in an index.

Data type: LOGICAL

Access: Readable

Applies to: Buffer-field object handle

KEYS attribute

Returns a comma-separated list of key fields for a buffer.

Data type: CHARACTER

Access: Readable

Applies to: Buffer object handle, Data-source object handle

For a Data-source object buffer, this attribute returns a comma-separated list of key fields defined in an associated KEYS clause for the specified buffer. If there are no defined key fields, this attribute returns a comma-separated list of key fields in the buffer's unique primary index (if any). If there are no defined key fields and no unique primary index, this attribute returns the string "ROWID".

Following is the syntax for accessing this attribute for a Data-source object buffer:

Syntax

```
data-source-handle:KEYS( buffer-sequence-number )
```

data-source-handle

The handle to the Data-source object.

buffer-sequence-number

An INTEGER that represents the sequence number of a buffer in the list of buffers for the Data-source object. Specify *buffer-sequence-number* to identify a buffer in the Data-source object when the Data-source object is defined against more than one database table buffer.

Note: Sequence numbers for buffers of a query start at one, where one represents the top level and subsequent numbers represent lower levels of join, if any.

Following is the syntax for accessing this attribute for a buffer directly:

Syntax

```
buffer-handle:KEYS
```

buffer-handle

The handle to the buffer.

LABEL attribute

The label of a widget or the name of a low-level event.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (column), Buffer-field object handle, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, MENU-ITEM widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, LAST-EVENT system handle

For the LAST-EVENT handle, this attribute is readable only.

For a widget, the LABEL attribute specifies the label for the widget. For some data representation widgets, it is actually the SCREEN-VALUE attribute value of the literal or text widget that is assigned to the SIDE-LABEL-HANDLE attribute of the specified data representation widget. For more information, see the SIDE-LABEL-HANDLE attribute reference entry.

For the LAST-EVENT handle, the LABEL attribute returns the names of low-level events based on the EVENT-TYPE attribute value. For EVENT-TYPE = "KEYPRESS", this attribute returns key label events, such as "F1" or "ESC". It also returns the key labels of any keys that trigger key function events (returned by the FUNCTION attribute).

For EVENT-TYPE = "MOUSE", this attribute returns low-level events for both portable and three-button mouse event types, such as "SELECT-MOUSE-UP" (portable) or "LEFT-MOUSE-UP" (three-button). It also returns the names of the low-level mouse actions that trigger any high-level mouse events (returned by the FUNCTION attribute).

For EVENT-TYPE = "PROGRESS", this attribute returns the same high-level event name returned by the FUNCTION attribute unless the Progress event is triggered by a key press. In this case, it returns the key label of the key that triggered the event.

Note: When the AUTO-RESIZE attribute is set to TRUE, Progress resizes button and toggle-box widgets with run-time changes to the LABEL attribute.

LABEL-BGCOLOR attribute

The color number of the background color for a column label or all column labels in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

LABEL-DCOLOR attribute (Character interfaces only)

The color number of the display color for a column label or all column labels in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

LABEL-FGCOLOR attribute

The color number of the foreground color for a column label or all column labels in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

LABEL-FONT attribute

The font for a column label or all column labels in a browse widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (column)

LABELS attribute

Indicates whether a label appears with the widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, TEXT widget

This attribute applies to static frames.

This attribute is writeable for the browse widget only. If the LABELS attribute is set to FALSE for a browse widget, no column headers will appear with the browse.

If the LABELS attribute is FALSE for a combo-box, editor, fill-in, or text widget, no label appears with the widget. If the attribute is FALSE for a frame, the NO-LABELS option is specified in the frame phrase for the frame and no labels are displayed with field-level widgets within the frame.

LANGUAGES attribute

A comma-separated list of all languages compiled into the r-code file specified by the RCODE-INFO:FILE-NAME attribute.

Data type: CHARACTER

Access: Readable

Applies to: RCODE-INFO system handle

If the r-code file was compiled with only the default language, LANGUAGES returns an empty string.

LARGE attribute (Windows only; Graphical interfaces only)

Indicates whether a Windows editor widget can hold over 20K of text. Non-Windows platforms ignore this attribute.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

When LARGE is FALSE, the Windows editor widget can hold a maximum of 20K of text. When LARGE is TRUE, the Windows editor widget can hold over 64K of text—the precise limit depends on available resources.

You can set this attribute only before the editor widget is realized.

Note: In character interfaces, the editor widget can hold large amounts of text by default. Therefore, character interfaces do not need separate “large” and “small” editors.

LARGE-TO-SMALL attribute

The default numeric range that a slider can display is small (minimum) to large (maximum). The LARGE-TO-SMALL option allows you to override this default behavior as follows:

- When the slider is positioned horizontally, the left-most position on the trackbar displays the maximum value and the right-most position displays the minimum value.
- When the slider is positioned vertically, the bottom-most position on the trackbar displays the maximum value and the top-most position displays the minimum value.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SLIDER widget

LAST-ASYNC-REQUEST attribute

Returns the last entry in the list of all current asynchronous request handles for the specified AppServer or Web service that have been created in the current session.

Data type: HANDLE

Access: Readable

Applies to: Server object handle

If there are no asynchronous request handles for the specified server, LAST-ASYNC-REQUEST returns the Unknown value (?).

LAST-BATCH attribute

Indicates whether a FILL operation on a ProDataSet temp-table retrieved the last batch of rows in its associated query.

Note: You typically use the LAST-BATCH attribute in conjunction with the BATCH-SIZE attribute. It is best to use the LAST-BATCH attribute with a top-level ProDataSet temp-table. If you use the LAST-BATCH attribute with the BATCH-SIZE attribute for a child temp-table, the parent (or ancestor) must have only one row, because Progress restarts the BATCH-SIZE counter on a child temp-table for each parent record (as opposed to once per child temp-table).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Temp-table object handle

See also BATCH-MODE attribute, BATCH-SIZE attribute, FILL( ) method

LAST-CHILD attribute

The handle of the last widget created in the container widget or the current session.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: DIALOG-BOX widget, FIELD-GROUP widget, FRAME widget, MENU widget, SESSION system handle, SUB-MENU widget, WINDOW widget

You can use the LAST-CHILD option to find the last entry in a list of all frames and dialog boxes in a window, all field groups in a frame or dialog box, all widgets in a field group, all menu items in a menu or submenu, or all windows in an OpenEdge session (SESSION handle). After finding the last entry, you can find the remaining entries in the list by using each widget's PREV-SIBLING attribute.

LAST-OBJECT attribute

The object reference for the last class object instance in the list of all valid instances created in the current OpenEdge session. If there are no class object instances in the current session, this attribute returns Unknown value (?).

Data type: Progress.Lang.Object

Access: Readable

Applies to: SESSION system handle

Once you get the last object reference in the list, you can use the PREV-SIBLING data member in the Progress.Lang.Object class to get the previous entry in the list of object references.

See also FIRST-OBJECT attribute, NEXT-SIBLING and PREV-SIBLING data members (in the Progress.Lang.Object class)

LAST-PROCEDURE attribute

For AppServer, returns a handle to the last entry in the list of remote persistent procedures running on the connected AppServer. For Web services, returns a handle to the last entry in the list of procedure objects associated with the Web service.

Data type: HANDLE

Access: Readable

Applies to: Server object handle, SESSION system handle

If the current session has no active persistent procedures or the AppServer has no active remote persistent procedures, LAST-PROCEDURE returns the Unknown value (?). To find the previous persistent procedure given the last, access the PREV-SIBLING attribute of the procedure handle you just got.

For information on creating persistent procedures, see the RUN statement reference entry. For more information on the AppServer, see *OpenEdge Application Server: Developing AppServer Applications*. To check a handle for validity, use the VALID-HANDLE function.

LAST-SERVER attribute

A handle to the last entry in the list of server handles for the current OpenEdge session. This includes both AppServer server objects and Web service server objects.

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

Returns the handle associated with the last entry in the list of all server handles created in the current session. If the current session has no server handles, LAST-SERVER has the Unknown value (?). For more information on server handles, see the Server object handle reference entry.

LAST-SERVER-SOCKET attribute

A handle to the last entry in the list of all valid server socket handles created in the current session. If there are no server socket handles in this session, LAST-SERVER-SOCKET returns the Unknown value (?).

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

LAST-SOCKET attribute

A handle to the last entry in the list of all valid socket handles created in the current session. If there are no socket handles in this session, LAST-SOCKET returns the Unknown value (?).

Data type: HANDLE

Access: Readable

Applies to: SESSION system handle

LAST-TAB-ITEM attribute

The last widget in the tab order of a field group.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: FIELD-GROUP widget

When you set this attribute, the assigned widget is moved to the last tab position, following the widget that was previously at this position. Other widgets in the field group maintain their same relative tab positions.

To set the attribute, you must assign it the widget handle of a field-level widget or frame that can receive focus from a TAB event and that is also a child of the field group to which the attribute applies. If the LAST-TAB-ITEM attribute is not set (is the Unknown value (?)), the default last tab position goes to the widget identified by the LAST-CHILD attribute of the field group.

For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry and *OpenEdge Development: Progress 4GL Handbook*.

Note: Any tab reordering that you do with this attribute can be reset by a subsequent ENABLE statement unless you define the frame that owns the field group with the KEEP-TAB-ORDER option. For more information, see the ENABLE statement and Frame phrase reference entries.

Left property (Windows only; Graphical interfaces only)

The horizontal position of the control-frame and control-frame COM object from the left side of the parent container widget, in pixels.

Return type: INTEGER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the COLUMN attribute and X attribute of the corresponding control-frame widget to an equivalent value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.

LENGTH attribute

The length (number of characters) of the current content of the editor widget.

Data type: INTEGER

Access: Readable

Applies to: EDITOR widget

If the editor widget is not already realized and you reference its LENGTH attribute, Progress realizes the widget.

In Windows, both the regular editor and the large editor support LENGTH.

LINE attribute

The current logical line number (iteration number) of the frame.

Data type: INTEGER

Access: Readable

Applies to: FRAME widget

This attribute applies to down frames only.

This attributes is equivalent to the FRAME-LINE function. For more information, see the FRAME-LINE function reference entry.

LIST-ITEM-PAIRS attribute

A list of the label-value pairs associated with a combo box or selection list. The list is delimiter-separated.

Note: The LIST-ITEM-PAIRS attribute applies only to combo boxes and selection lists whose entries consist of label-value pairs. For combo boxes and selection lists whose entries consist of single items, use the LIST-ITEMS attribute.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: COMBO-BOX widget, SELECTION-LIST widget

The value of the delimiter depends on the value of the DELIMITER attribute, which is comma by default.

LIST-ITEM-PAIRS provides a list like the following:

```
"Red,1,Blue,2,Green,3"
```

LIST-ITEMS attribute

A list of the items associated with a combo box or selection list. The list is delimiter-separated.

Note: The LIST-ITEMS attribute applies only to combo boxes and selection lists whose entries consist of single items. For combo boxes and selection lists whose entries consist of label-value pairs, use the LIST-ITEM-PAIRS attribute.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: COMBO-BOX widget, SELECTION-LIST widget

The value of the delimiter depends on the value of the DELIMITER attribute, which is comma by default.

LIST-ITEMS provides a list like the following:

```
"Red,Blue,Green"
```

LIST-PROPERTY-NAMES( ) method

Returns a comma-separated list of all application-defined properties associated with the Client-principal object. The Client-principal object may be sealed or unsealed.

Return type: CHARACTER

Applies to: Client-principal object handle

Syntax

```
LIST-PROPERTY-NAMES( )
```

You can also use the GET-PROPERTY( ) method to get the value of a single property associated with a Client-principal object.

Example The following code fragment illustrates how to use the LIST-PROPERTY-NAMES( ) method:

```
DEF VAR hCP as HANDLE.
DEF VAR vProp as CHAR.
.
.
.
CREATE CLIENT-PRINCIPAL hCp.
.
.
.
vProp = hCP:LIST-PROPERTY-NAMES( ).
DISPLAY vProp format “X(70)”.
```

See also CAN-DO function, ENTRY function, GET-PROPERTY( ) method, NUM-ENTRIES function, SET-PROPERTY( ) method

LITERAL-QUESTION attribute

Lets you specify how Progress interprets a quoted character value during assignment into the BUFFER-VALUE attribute for a character BUFFER-FIELD object. That is, whether Progress treats the quoted character value as a literal or non-literal character value.

Return type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer-field object handle

When TRUE, Progress treats a quoted character value as a literal character value. That is, it does not remove enclosing quotes, trailing blanks, or formatting insertion characters.

When FALSE, the default value, Progress treats a quoted character value as a non-literal character value. That is, it removes enclosing quotes, trailing blanks, and formatting insertion characters. For example:

- Progress treats "`abc  `" as `abc`.
- Progress treats a quoted question mark character ("?") as the Unknown value (`?`).

You can use the Literal Question (`-literalquestion`) startup parameter to change the default value of the LITERAL-QUESTION attribute to TRUE (which would otherwise be FALSE). For more information about this startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

See also BUFFER-VALUE attribute

LOAD( ) method

Loads an XML document into memory, parses it, and makes its contents available in the 4GL.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
LOAD( mode , { file | memptr | longchar } , validate )
```

mode

A character expression that evaluates to one of the following: “FILE”, “MEMPTR”, or “LONGCHAR”.

file

A character expression that represents the name of a file. You can specify a relative pathname, an absolute pathname, or an HTTP URL.

memptr

A MEMPTR variable that contains the loaded XML text. The size of the MEMPTR variable should match the size of the XML text.

longchar

A LONGCHAR variable that contains the loaded XML text. The size of the LONGCHAR variable should match the size of the XML text.

validate

A logical expression where TRUE indicates that the parser should validate the document’s logical structure with respect to its Document Type Definition (DTD). Note that even if validation against the DTD is not specified, the document’s physical structure is still validated. If this expression is TRUE, then the parser will also validate against any XML Schema file references in the XML document or specified in the SCHEMA-LOCATION and NONAMESPACE-SCHEMA-LOCATION attribute.

Note This method sets the ENCODING attribute for the XML document to the encoding name specified in the XML document’s encoding declaration.

Example The following code fragment creates a parse tree of XML nodes and validates its structure:

```
DEFINE VARIABLE hDoc AS HANDLE.
CREATE X-DOCUMENT hDoc.
hDoc:LOAD("file","memo.xml",true).
. . .
```

See also ADD-SCHEMA-LOCATION( ) method, NONAMESPACE-SCHEMA-LOCATION attribute, SCHEMA-LOCATION attribute, SCHEMA-PATH attribute

LoadControls( ) method (Windows only; Graphical interfaces only)

Loads the control from a specified control file into the specified control-frame.

Return type: None

Applies to: CONTROL-FRAME widget, COM object

Syntax

```
LoadControls ( control-filename , control-frame-name )
```

control-filename

The name and extension of a control (`.wrx`) file associated with the current external procedure that is created by the AppBuilder at design time.

control-frame-name

A character-string expression that specifies the section of the control file that contains the control. Typically, this section name is also the name of the control-frame defined by the AppBuilder at design time.

This method loads the specified control along with all of its design-time property values.

Notes

- References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.
- In a future release, this method loads multiple controls into a control-frame.

LOAD-DOMAINS( ) method

Loads registered authentication domains from the specified OpenEdge database into the application's trusted authentication domain registry. Progress uses this registry to validate Client-principal objects for the application during the session. After loading the registered domains, Progress automatically restricts the registration of additional domains for the remainder of the session.

Note: Using this method to load registered authentication domains directly from an OpenEdge database provides more security than using the REGISTER-DOMAIN( ) and LOCK-REGISTRATION( ) methods because it automatically locks the registry.

Return type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
LOAD-DOMAINS( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of a connected database from which to load registered domains. For example, LOAD-DOMAINS(1) loads registered domains from the first database, LOAD-DOMAINS(2) loads registered domains from the second database, and so on. If you specify a sequence number that does not correspond to a connected database, Progress generates a run-time error.

logical-name or *alias*

The logical name or alias of a connected database from which to load registered domains. These forms require a quoted character string or a character expression. If you specify a logical name or alias that does not correspond to a connected database, Progress generates a run-time error.

You can call this method only once per session.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

See also LOCK-REGISTRATION( ) method, REGISTER-DOMAIN( ) method

LOAD-ICON( ) method

Specifies the file that contains the icon that you want to load for display in the title bar of a window, in the task bar, and when selecting a program using **ALT+TAB**. This method can accommodate icons formatted as small size (16x16) icons, regular size (32x32) icons, or both.

An icon file might contain multiple icons. In those instances when multiple icons are in a file, this method uses the 32x32 icon, if one exists, from the file that you specified. However, if a 32x32 icon does not exist, it uses the first icon in the file.

You must use this method if you want a specific program icon to display when selecting a program using **ALT+TAB**.

If the load is successful, this method returns TRUE.

Return type: LOGICAL

Applies to: WINDOW widget

Syntax

```
LOAD-ICON ( icon-filename [ , n ] )
```

icon-filename

A character-string expression that specifies a full or relative pathname for a file that contains the icon that you want to load for display in the title bar of a window, in the task bar, and when selecting a program using **ALT+TAB**.

n

An integer expression that specifies the position of the icon within the file. Only use this expression if you want to override the default behavior.

For example, LOAD-ICON(“file.ico”, 2) finds the second icon in the icon file `file.ico` and loads it. LOAD-ICON(“”) removes the previously loaded icon.

Notes

- In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-ICON loads the icon file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-ICON method continues searching with the next PROPATH entry.

- If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-ICON method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the icon files. Use this value to create a fully-qualified URL pathname for *icon-filename* and avoid repeated searches of the PROPATH.

LOAD-IMAGE( ) method

Reads the image contained in a specified file. When applied to a button widget, the image is used for the button in its up state, and also for its down state if a separate down state image is not specified. For buttons, this is equivalent to the LOAD-IMAGE-UP( ) method.

Return type: LOGICAL

Applies to: BUTTON widget, IMAGE widget

Syntax

```
LOAD-IMAGE ( filename
             [ ,
                  x-offset ,
                  y-offset ,
                  width ,
                  height
             ] )
```

filename

A character-string expression that specifies a full or relative pathname for a file that contains an image.

x-offset

An integer expression that specifies the pixel along the x-axis at which to begin reading from the image file.

y-offset

An integer expression that specifies the pixel along the y-axis at which to begin reading from the image file.

width

An integer expression that specifies the number of pixels along the x-axis to read from the image file.

height

An integer expression that specifies the number of pixels along the y-axis to read from the image file.

Notes

- The image is not displayed until the widget is realized. If the read is successful, the method returns TRUE.
- In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-IMAGE loads the image file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-IMAGE method continues searching with the next PROPATH entry.

- If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-IMAGE method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value to create a fully-qualified URL pathname for *filename* and avoid repeated searches of the PROPATH.

LOAD-IMAGE-DOWN( ) method

Reads the image contained in a specified file. The image is used for the button in its down state only.

Return type: LOGICAL

Applies to: BUTTON widget

Syntax

```
LOAD-IMAGE-DOWN ( filename
                  [ ,
                      x-offset ,
                      y-offset ,
                      width ,
                      height
                  ] )
```

filename

A character-string expression that specifies a full or relative pathname for a file that contains an image to display in a button when the button is in its down state.

x-offset

An integer expression that specifies the pixel along the x-axis at which to begin reading from the image file.

y-offset

An integer expression that specifies the pixel along the y-axis at which to begin reading from the image file.

width

An integer expression that specifies the number of pixels along the x-axis to read from the image file.

height

An integer expression that specifies the number of pixels along the y-axis to read from the image file.

Notes

- The image is not displayed until the widget is realized. If the read is successful, the method returns TRUE.
- In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-IMAGE-DOWN loads the image file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-IMAGE-DOWN method continues searching with the next PROPATH entry.

- If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-IMAGE-DOWN method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value to create a fully-qualified URL pathname for *filename* and avoid repeated searches of the PROPATH.

LOAD-IMAGE-INSENSITIVE( ) method

Reads the image contained in specified file. The image is used for the button in its insensitive state.

Return type: LOGICAL

Applies to: BUTTON widget

Syntax

```
LOAD-IMAGE-INSENSITIVE ( filename
                         [ ,
                             x-offset ,
                             y-offset ,
                             width ,
                             height
                         ] )
```

filename

A character-string expression that specifies a full or relative pathname for a file that contains an image to display in a button when the button is insensitive.

x-offset

An integer expression that specifies the pixel along the x-axis at which to begin reading from the image file.

y-offset

An integer expression that specifies the pixel along the y-axis at which to begin reading from the image file.

width

An integer expression that specifies the number of pixels along the x-axis to read from the image file.

height

An integer expression that specifies the number of pixels along the y-axis to read from the image file.

Notes

- If this image is not loaded, the appearance of the button up image does not change between the button's sensitive and insensitive state; if no image is loaded, the button label is grayed out in its insensitive state.
- The image is not displayed until the widget is realized. If the read is successful, the method returns TRUE.
- In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-IMAGE-INSENSITIVE loads the image file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-IMAGE-INSENSITIVE method continues searching with the next PROPATH entry.

- If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-IMAGE-INSENSITIVE method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value to create a fully-qualified URL pathname for *filename* and avoid repeated searches of the PROPATH.

LOAD-IMAGE-UP( ) method

Reads the image contained in a specified file. The image is used for the button in its up state. The image is also used for the down state if a separate down image is not specified. This method is equivalent to the LOAD-IMAGE( ) method.

Return type: LOGICAL

Applies to: BUTTON widget

Syntax

```
LOAD-IMAGE-UP ( filename
                [ ,
                    x-offset ,
                    y-offset ,
                    width ,
                    height
                ] )
```

filename

A character-string expression that specifies a full or relative pathname for a file that contains an image to display in a button when the button is in its up state.

x-offset

An integer expression that specifies the pixel along the x-axis at which to begin reading from the image file.

y-offset

An integer expression that specifies the pixel along the y-axis at which to begin reading from the image file.

width

An integer expression that specifies the number of pixels along the x-axis to read from the image file.

height

An integer expression that specifies the number of pixels along the y-axis to read from the image file.

Notes

- The image is not displayed until the button is realized. If the read is successful, the method returns TRUE.
- In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-IMAGE-UP loads the image file directly without searching directories or URLs in PROPATH. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the PROPATH, the LOAD-IMAGE-UP method continues searching with the next PROPATH entry.

- If you specify URL pathnames on the PROPATH and your application repeatedly uses the LOAD-IMAGE-UP method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the image files. Use this value to create a fully-qualified URL pathname for *filename* and avoid repeated searches of the PROPATH.

LOAD-MOUSE-POINTER( ) method (Graphical interfaces only)

Specifies the mouse pointer to display when the pointer is moved over the widget. If you apply this method to a frame, field group, or window, the same mouse pointer is displayed when it is moved across all child widgets within the frame, field group, or window. However, if you load a different mouse pointer for a child widget, the child widget mouse pointer is displayed when it is moved over that child.

Return type: LOGICAL

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
LOAD-MOUSE-POINTER ( pointer-name )
```

pointer-name

A character-string expression that specifies the name of a mouse pointer.

Progress provides a collection of mouse pointers that you can use in graphical applications. Table 71 names and describes each mouse pointer in the collection.

Table 71: Progress mouse pointers

Pointer name	Description
APPSTARTING	Arrow with an hourglass beside it
ARROW	Standard arrow cursor
CROSS	Cross hairs
HELP	Arrow with a question mark beside it
IBEAM	I-beam text cursor
NO	Circle with a slash through it
RECTANGLE	(NT 3.51 only) White rectangle
SIZE	Sizing rectangle
SIZE-E	Size to right
SIZE-N	Size to top
SIZE-NE	Size to top right
SIZE-NW	Size to top left
SIZE-S	Size to bottom
SIZE-SE	Size to bottom right
SIZE-SW	Size to bottom left
SIZE-W	Size to left
UPARROW	Up arrow
WAIT	System busy
GLOVE	Glove/finger
COMPILER-WAIT	Compiler busy

Notes

- If the mouse pointer is loaded successfully, the method returns TRUE.
- In addition to the mouse pointers that Progress supplies, you can also use a bitmap that you supply that is in the form of a Windows cursor (`.cur` or `.ani`) file. To use such a bitmap, substitute the name of the Windows cursor file for *`pointer-name`*.
- For browse-columns, if you do not specify a mouse pointer, Progress uses the mouse pointer the user specified for the browse.
- In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-MOUSE-POINTER loads the pointer file directly without searching directories or URLs in `PROPATH`. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the `PROPATH`, the LOAD-MOUSE-POINTER method continues searching with the next `PROPATH` entry.

- If you specify URL pathnames on the `PROPATH` and your application repeatedly uses the LOAD-MOUSE-POINTER method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the pointer files. Use this value to create a fully-qualified URL pathname for *`pointer-name`* and avoid repeated searches of the `PROPATH`.

LOAD-SMALL-ICON( ) method (Windows only; Graphical interfaces only)

Specifies the file that contains the icon that you want to load for display in the title bar of a window and in the task bar only. This method can accommodate icons formatted as small size (16x16) icons, regular size (32x32) icons, or both.

The icon file might contain multiple icons. In those instances when multiple icons are in a file, the LOAD-SMALL-ICON method, by default, uses the 16x16 icon, if one exists, from the file that you specified. Otherwise, it uses the first icon in the file. If it uses a 32x32 icon, it reduces its size to a 16x16 format in both the title bar and the task bar.

If the load is successful, this method returns TRUE.

Note: You cannot use this method to display a specific icon when selecting a program using **ALT+TAB**; you must use the LOAD-ICON( ) method for this purpose. Otherwise, Progress displays the default icon.

Return type: LOGICAL

Applies to: WINDOW widget

Syntax

```
LOAD-SMALL-ICON ( smallicon-filename [ n ] )
```

smallicon-filename

A character-string expression that specifies the name of a file that contains the icon you want to load for display in the title bar of a window and in the task bar.

n

An integer expression that specifies the position of an icon within the file. Only use this expression if you want to override the default behavior.

Notes

- For example, LOAD-SMALL-ICON("file.ico", 2) finds the second icon in the icon file `file.ico` and loads it. LOAD-SMALL-ICON("") removes the previously loaded icon.
- The LOAD-SMALL-ICON method is only available in Windows 95 and NT Version 4.0. If you try to use it with any other platform, this method returns FALSE.
- In Windows, you can specify a URL pathname. If you specify a fully-qualified URL, LOAD-SMALL-ICON loads the icon file directly without searching directories or URLs in `PROPATH`. Valid URL protocols include HTTP and HTTPS.

  Note: URL pathnames cannot contain the percent symbol (%). If an error exists in a URL specified on the `PROPATH`, the LOAD-SMALL-ICON method continues searching with the next `PROPATH` entry.

- If you specify URL pathnames on the `PROPATH` and your application repeatedly uses the LOAD-SMALL-ICON method with a URL pathname, you can improve performance by using the SEARCH function once to determine the full URL pathname to the directory containing the icon files. Use this value to create a fully-qualified URL pathname for *`smallicon-filename`* and avoid repeated searches of the `PROPATH`.

LOCAL-HOST attribute

Indicates the IP (Internet Protocol) address of the machine the socket object is communicating with.

Data type: CHARACTER

Access: Readable

Applies to: Socket object handle

When a server and client successfully establish a connection, both the server and client have a socket object that identifies this connection. On the client and on the server, this attribute returns the IP address of the machine which is making the request. If the CONNECT method failed, this attribute returns the Unknown value (?).

LOCAL-NAME attribute

This attribute returns the unqualified part of a namespace-aware XML node name or SOAP-header-entryref element name (that is, the part after the colon character). For nodes created with the CREATE-NODE( ) method, or nodes of any type other than ELEMENT or ATTRIBUTE, this attribute returns " ".

Data type: CHARACTER

Access: Readable

Applies to: SOAP-header-entryref object handle, X-noderef object handle

LOCAL-PORT attribute

Indicates the port number of the socket.

Data type: INTEGER

Access: Readable

Applies to: Socket object handle

When a server and client successfully establish a connection, both the server and client have a socket object that identifies this connection. On the client, this attribute returns the port number used on the client machine for this socket connection. On the server, this attribute returns the port number used on the server machine for this socket connection. If the CONNECT failed, this attribute returns the Unknown value (?).

LOCATOR-COLUMN-NUMBER attribute

The current column in the XML source.

Data type: INTEGER

Access: Readable

Applies to: SAX-reader object handle

Valid only in a callback. Use the SELF handle to get it, as in the following fragment:

```
myColNum = SELF:LOCATOR-COLUMN-NUMBER.
```

Gives the column number where the text that caused the current callback ends. The first column in a line is 1.

If the current location is an **external entity**—that is, external to the main XML source—the column number is relative to the beginning of the line in the external entity.

LOCATOR-LINE-NUMBER attribute

The current line in the XML source.

Data type: INTEGER

Access: Readable

Applies to: SAX-reader object handle

Valid only in a callback. Use the SELF handle to get it, as in the following fragment:

```
myLineNum = SELF:LOCATOR-LINE-NUMBER.
```

Gives the line number where the text that caused the current callback ends. The first line in a document is 1.

If the current location is an **external entity**—that is, external to the main XML source—the line number is relative to the beginning of the external entity.

LOCATOR-PUBLIC-ID attribute

Returns the public identifier of the current XML source.

Data type: CHARACTER

Access: Readable

Applies to: SAX-reader object handle

Valid only in a callback. Use the SELF handle to get the public identifier of the XML source, as in the following fragment:

```
myPublicID = SELF:LOCATOR-PUBLIC-ID.
```

LOCATOR-SYSTEM-ID attribute

Returns the system identifier of the current XML source.

Data type: CHARACTER

Access: Readable

Applies to: SAX-reader object handle

Valid only in a callback. Use the SELF handle to get the system identifier of the XML source, as in the following fragment:

```
mySystemID = SELF:LOCATOR-SYSTEM-ID.
```

LOCATOR-TYPE attribute

The type of server on which the application files are stored.

Data type: CHARACTER

Access: Readable

Applies to: CODEBASE-LOCATOR system handle

Valid values are "AppServer" or "InternetServer".

LOCKED attribute

Indicates whether another user has a lock on a record that a GET . . . WAIT statement or method is trying to access.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle

Note: The LOCKED attribute corresponds to the LOCKED function.

LOCK-REGISTRATION( ) method

Restricts the registration of additional authentication domains in the application's trusted authentication domain registry for the remainder of the session. You must call this method after you have registered all authentication domains for a session using the REGISTER-DOMAIN( ) method.

Return type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
LOCK-REGISTRATION( )
```

You must call this method before you can use the trusted authentication domain registry to validate Client-principal objects for the application.

If you do not register at least one authentication domain in the trusted authentication domain registry before calling this method, this method returns TRUE. However, any attempt to seal a Client-principal object will generate a run-time error.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

See also LOAD-DOMAINS( ) method, REGISTER-DOMAIN( ) method

LOG-AUDIT-EVENT( ) method

Creates an audit record for the specified application-defined audit event in each connected audit-enabled database whose current audit policy has this audit event enabled.

This method returns a Base64 character string that specifies a universally unique identifier (UUID) as the primary index for the generated audit event record. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Return type: CHARACTER

Applies to: AUDIT-CONTROL system handle

Syntax

```
LOG-AUDIT-EVENT( event-id, event-context [, event-detail
  [, user-detail ] ] )
```

event-id

An integer value that specifies an identifier for an application-defined audit event. This value must be greater than or equal to 32000.

event-context

A character expression that specifies the context for the audit event. The value of this expression cannot exceed 200 characters. You can also use this value as an alternate index for querying the audit event record.

If you specify the Unknown value (?), Progress generates a run-time error.

event-detail

An optional character expression that specifies additional audit detail. The value of this expression cannot exceed 10,000 characters.

user-detail

An optional character expression that specifies additional user detail. The value of this expression cannot exceed 10,000 characters.

LOG-ENTRY-TYPES attribute

A comma-separated list of one or more types of log entries to write to the log file.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: LOG-MANAGER system handle

Use the LOG-ENTRY-TYPES attribute to specify one or more types of log entries to write to the log file specified by the LOGFILE-NAME attribute or the Client Logging (`-clientlog`) startup parameter.

The LOG-ENTRY-TYPES attribute corresponds to the Log Entry Types (`-logentrytypes`) startup parameter.

By default, the logging level you specify using the LOGGING-LEVEL attribute or the Logging Level (`-logginglevel`) startup parameter applies to all log entry types specified. However, you can specify a different logging level for each entry type, as follows:

```
LOG-MANAGER:LOG-ENTRY-TYPES = log-entry-type[:level]
```

log-entry-type

A log entry type listed in Table 72 below.

level

A logging level value (between 2 and 4).

Table 72 describes the log entry types.

Table 72: Log entry types *(1 of 3)*

Log entry type	Executables	Description
`4GLMessages`	4GL (GUI and character mode). AppServer and WebSpeed do not require this log entry type for 4GL messages to be written to the log file.	The 4GL interpreter writes all 4GL VIEW-AS ALERT-BOX messages to the log file, together with the 4GL stack, when you turn on Debug Alert using either the Debug Alert (`-debugalert`) startup parameter or the DEBUG-ALERT attribute on the SESSION system handle.
`4GLTrace`	4GL clients, AppServer, and WebSpeed agents	Turns on logging for the execution of internal procedures, user-defined functions, persistent user-interface triggers, and named events (generated by the RUN, FUNCTION, PUBLISH, and SUBSCRIBE statements, respectively). It also logs the instantiation of classes (generated by the NEW statement) and the invocation of methods defined within classes (generated by the SUPER( ) method and DELETE OBJECT statement).

`AiAMgmt`
`AiaProp`
`AiaRqst`
`AiaUbroker`
`AiaDefault`
	AIA	Turns on logging for the AIA component.

`ASDefault`	AppServer agent	Combines the `AsPlumbing` and `DB.Connects` log entry types. It is the default value for AppServer agents.
`ASPlumbing`	AppServer agent	Turns on logging for different actions, depending on the logging level specified.

Table 72: Log entry types *(2 of 3)*

Log entry type	Executables	Description
DB.Connects	4GL clients, AppServer, and WebSpeed agents	Turns on logging of database connections and disconnections. The log messages include database name and user ID number.
DynObjects.Class DynObjects.DB DynObjects.XML DynObjects.Other DynObjects.UI	4GL clients, AppServer, and WebSpeed agents 4GL clients and WebSpeed agents	Turns on logging for the creation and deletion of database objects, dynamic objects, user interface objects, and classes. For a list of objects within each category and a description of the log entries, see *OpenEdge Development: Debugging and Troubleshooting*.
FileID	4GL clients, AppServer, and WebSpeed agents	Turns on logging for file opening, file closing, and error messages that do not contain the file name.
MsgrTrace	WebSpeed Messengers	Turns on logging for WebSpeed Messengers. The information logged depends on which Messenger is running and the logging level specified.
NSPlumbing	NameServer	Turns on logging for the NameServer component.
ProEvents.UI.Char ProEvents.UI.Command ProEvents.Other	4GL clients, AppServer, and WebSpeed agents	Turns on logging for different categories of events. For a list of events within each category and a description of the log entries, see *OpenEdge Development: Debugging and Troubleshooting*.
QryInfo	4GL clients, AppServer, and WebSpeed agents	Turns on logging for query resolution statistics.
SAX	4GL clients, AppServer, and WebSpeed agents	Turns on logging for the SAX parser.

Table 72: Log entry types *(3 of 3)*

Log entry type	Executables	Description
UBroker.Basic UBroker.ClientFSM UBroker.ServerFSM UBroker.ClientMsgStream UBroker.ServerMsgStream UBroker.ClientMsgQueue UBroker.ServerMsgQueue UBroker.ClientMemTrace UBroker.ServerMemTrace UBroker.ThreadPool UBroker.Stats UBroker.AutoTrim UBroker.All	Unified Broker	Turns on logging for the Unified Broker component.
WSADefault	Web Services Adapter (WSA)	Turns on logging for the Web Services Adapter component.

The following example shows how to specify one or more log entry types:

```
LOG-MANAGER:LOG-ENTRY-TYPES = "DB.Connects,4GLTrace:2,DynObjects.UI:3"
```

The following example shows how to specify all log entry types within a category:

```
LOG-MANAGER:LOG-ENTRY-TYPES = "DynObjects.*"
```

You can also turn off logging by setting this attribute to the Unknown value (`?`).

See also the reference entries for the Log Entry Types (`-logentrytypes`), Client Logging (`-clientlog`), and Logging Level (`-logginglevel`) startup parameters in *OpenEdge Deployment: Startup Command and Parameter Reference*.

For more information about log entry types and logging levels, see *OpenEdge Development: Debugging and Troubleshooting*.

LOG-THRESHOLD attribute

The file size threshold of log files. When the current log file becomes equal to or greater than the specified size, Progress renames and saves the log file and creates a new log file.

Data type: INTEGER

Access: Readable

Applies to: LOG-MANAGER system handle

Valid values are:

- **0** — This means there is no limit other than what the operating system imposes. Specify 0 to ignore the Number of Log Files to Keep (`-numlogfiles`) startup parameter setting. This is the default.
- **Between 500,000 and 2,147,483,647** — Values are in bytes (one byte typically holds one character). You can specify a file size up to 2GB, inclusive, but not lower than 500,000.

Progress names log files based on a sequence number using the following format:

```
<filename>.999999.<extension>
```

For example, if you specify a log file named `my.log`, Progress renames the log file to `my.000001.log` before creating a new log file.

The LOG-THRESHOLD attribute corresponds to the Log Threshold (`-logthreshold`) startup parameter.

Use the LOGFILE-NAME attribute or the Client Logging (`-clientlog`) startup parameter to specify a log file name for 4GL clients. Use the NUM-LOG-FILES attribute or the Number of Log Files to Keep (`-numlogfiles`) startup parameter to specify the number of log files to keep.

For more information about the Client Logging (`-clientlog`) and Number of Log Files to Keep (`-numlogfiles`) startup parameters, see *OpenEdge Deployment: Startup Command and Parameter Reference* and *OpenEdge Development: Debugging and Troubleshooting*.

LOGFILE-NAME attribute

The name of log file OpenEdge uses to log messages and 4GL stack trace information.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: LOG-MANAGER system handle

If the filename you supply is a relative pathname, then a file is accessed relative to the current working directory. If the filename is an absolute pathname, then the specified file is accessed.

Note: Do not include a numbered sequence in the filename. This might conflict with the rolled over log files OpenEdge creates based on your NUM-LOG-FILES attribute and LOG-THRESHOLD attribute settings.

When the specified log file is open, this attribute is read-only.

The LOGFILE-NAME attribute corresponds to the Client Logging (`-clientlog`) startup parameter.

Use the LOG-ENTRY-TYPES attribute or the Log Entry Types (`-logentrytypes`) startup parameter to specify one or more types of log entries you want to write to the log file. Use the LOGGING-LEVEL attribute or the Logging Level (`-logginglevel`) startup parameter to specify the level at which log entries are written to the log file.

For more information about the Log Entry Types (`-logentrytypes`) and Logging Level (`-logginglevel`) startup parameters, see *OpenEdge Deployment: Startup Command and Parameter Reference* and *OpenEdge Development: Debugging and Troubleshooting*.

LOGGING-LEVEL attribute

The level at which log entries are written to the log file.

Data type: INTEGER

Access: Readable/Writeable

Applies to: LOG-MANAGER system handle

Use the LOGGING-LEVEL attribute to specify the level at which log entries are written to the log file specified by the LOGFILE-NAME attribute or the Client Logging (`-clientlog`) startup parameter. Each logging level specifies a different amount of information.

There are five logging levels:

- **0 (None)** — Log no entries. This is equivalent to turning logging off.
- **1 (Errors)** — Log Progress error messages. This includes all error messages and is unrelated to the entry types specified. Errors are logged at level 1 (Errors) and higher.
- **2 (Basic)** — Log entry type determines the logged information.
- **3 (Verbose)** — Log entry type determines the logged information.
- **4 (Extended)** — Log entry type determines the logged information.

By default, the logging level you specify applies to all log entry types. However, you can specify a different logging level for individual log entry types with using the LOG-ENTRY-TYPES attribute or the Log Entry Types (`-logentrytypes`) startup parameter.

The LOGGING-LEVEL attribute corresponds to the Logging Level (`-logginglevel`) startup parameter.

For more information about the Log Entry Types (`-logentrytypes`), Client Logging (`-clientlog`), or Logging Level (`-logginglevel`) startup parameters, see *OpenEdge Deployment: Startup Command and Parameter Reference* and *OpenEdge Development: Debugging and Troubleshooting*.

LOGIN-EXPIRATION-TIMESTAMP attribute

The time stamp of when the Client-principal object will expire. If the Client-principal object expires before you can seal it or validate it, Progress sets the LOGIN-STATE attribute to "EXPIRED" and you can no longer validate or use the Client-principal object.

Data type: DATETIME-TZ

Access: Readable/Writeable

Applies to: Client-principal object handle

If not specified, Progress will not place the Client-principal object in an EXPIRED login state and will not check the object for the EXPIRED login state condition. In this case, Progress returns the Unknown value (?).

Note: Progress recognizes that a Client-principal object has expired only when it tries to use it with the SET-CLIENT( ) method or SET-DB-CLIENT function.

Once the Client-principal object is sealed, this attribute is read-only.

See also LOGIN-STATE attribute

LOGIN-HOST attribute

The name of the host system on which the user represented by the Client-principal object was authenticated. If not specified, Progress returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the Client-principal object is sealed, this attribute is read-only.

LOGIN-STATE attribute

Returns a character value that represents the current state of the Client-principal object. Valid values are: “LOGIN”, “LOGOUT”, “EXPIRED”, and “FAILED”. The default value is “LOGOUT”.

Data type: CHARACTER

Access: Readable

Applies to: Client-principal object handle

Progress also sets the STATE-DETAIL attribute with a description of the current state.

See also LOGIN-EXPIRATION-TIMESTAMP attribute, STATE-DETAIL attribute

LOGOUT( ) method

Indicates that the user represented by the sealed Client-principal object has logged out of the current user login session. This is a general purpose method an application can use to invalidate, or terminate access to, a sealed Client-principal object.

Once logged out, you can no longer use the Client-principal object to set a user ID using either the SET-CLIENT( ) method or SET-DB-CLIENT function.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
LOGOUT( )
```

If you call this method for an unsealed Client-principal object, Progress generates a run-time error. You can use the AUTHENTICATION-FAILED( ) method to invalidate an **unsealed** Client-principal object.

Progress also sets the LOGIN-STATE attribute for the Client-principal object to “LOGOUT”.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

Calling this method generates an audit event and creates an audit record for the event in all connected audit-enabled databases according to each database’s current audit policy settings.

See also AUTHENTICATION-FAILED( ) method, LOGIN-STATE attribute, SET-CLIENT( ) method, SET-DB-CLIENT function

LONGCHAR-TO-NODE-VALUE( ) method

Sets the value of an X-noderef node to the contents of a LONGCHAR.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
LONGCHAR-TO-NODE-VALUE( longchar )
```

longchar

An expression of type LONGCHAR.

LONGCHAR-TO-NODE-VALUE( ) raises an error if any of the following occurs:

- The node is read only.
- The node contains invalid XML characters.
- The data in *longchar* is not null-terminated.
- *longchar* is not set to the exact size of the valid data.

For more information on accessing XML documents using the Document Object Model (DOM) interface, see *OpenEdge Development: Programming Interfaces*.

LOOKUP( ) method

Returns the index of the specified item in a combo-box list or selection list.

Return type: INTEGER

Applies to: COMBO-BOX widget, SELECTION-LIST widget

Syntax

```
LOOKUP ( list-string )
```

list-string

A character-string expression that specifies a single value in the combo box or selection list.

If *list-string* has the Unknown value (?), LOOKUP returns the Unknown value (?). If *list-string* is not in the list, LOOKUP returns 0.

MANDATORY attribute

Indicates whether a buffer-field is a required field.

Data type: LOGICAL

Access: Readable

Applies to: Buffer-field object handle

MANUAL-HIGHLIGHT attribute

Indicates whether a widget exhibits custom or standard highlight behavior when selected.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Set the MANUAL-HIGHLIGHT attribute to TRUE to use a customized highlight design for selection of the widget. A FALSE value for this attribute specifies the Progress default highlight behavior for the selection of the widget. For more information, see *OpenEdge Development: Progress 4GL Handbook*.

MAX-BUTTON attribute (Windows only; Graphical interfaces only)

Determines whether the window has a maximize button in its caption bar.

Data type: LOGICAL

Access: Readable/Writable

Applies to: WINDOW widget

In character interfaces, this attribute has no effect.

In Windows, a window can have maximize and minimize buttons depending on the settings of the MAX-BUTTON and MIN-BUTTON attributes. Both buttons are created on the window. If you set the MAX-BUTTON to TRUE and the MIN-BUTTON to FALSE, only the maximize button is enabled; the minimize button is disabled.

The MAX-BUTTON attribute must be set before the window is realized. The default value is TRUE.

MAX-CHARS attribute (Graphical interfaces only)

The maximum number of characters an editor or combo-box widget can hold.

Data type: INTEGER

Access: Readable/Writeable

Applies to: COMBO-BOX widget, EDITOR widget

For editor widgets, you can set this attribute only before the widget is realized. In Windows, the maximum value of MAX-CHARS is approximately 20K for the regular editor and over 64K for the large editor.

For SIMPLE and DROP-DOWN combo-box widgets, you can set this attribute before or after the widget is realized. If the value of MAX-CHARS for a combo-box widget is zero or the Unknown value (?), the default value is 255 characters. This attribute is ignored for DROP-DOWN-LIST combo-box widgets.

Note: In character interfaces, editors can grow until Progress runs out of system resources.

MAX-DATA-GUESS attribute

The estimated number of records in a browse query.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget

Before enabling the browse widget, set this attribute to the exact or maximum number of records you expect in the query. A more accurate setting of this attribute allows for a smoother and more accurate change in vertical thumb height when the user scrolls through the query for the first time. As a user scrolls through the records, the system continuously updates the value of this attribute with a better guess for the number of records. After all records have been read, the MAX-DATA-GUESS value is automatically reset to the exact number for more accurate browsing. The default value is 100.

MAX-HEIGHT-CHARS attribute

The maximum height of the window, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: WINDOW widget

MAX-HEIGHT-PIXELS attribute

The maximum height of the window, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

MAX-VALUE attribute

The maximum value for a slider.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SLIDER widget

You can set this attribute only before the widget is realized.

MAX-WIDTH-CHARS attribute

The maximum width of a window, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: WINDOW widget

MAX-WIDTH-PIXELS attribute

The maximum width of a window, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

MD5-VALUE attribute

Returns the MD5 value stored in an r-code file.

Data type: CHARACTER

Access: Readable

Applies to: RCODE-INFO system handle

The return value is a 32 character hexadecimal number.

Use this attribute to determine if a procedure changed between different versions of your application.

If you did not use the GENERATE-MD5 option on the COMPILE statement to compile a procedure or class, Progress did not store the MD5 value in the r-code file. In this case, this attribute returns the Unknown value (?).

MEMPTR-TO-NODE-VALUE( ) method

Sets the value of an X-noderef node to the contents of a MEMPTR.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
MEMPTR-TO-NODE-VALUE( memptr )
```

memptr

An expression of type MEMPTR.

MEMPTR-TO-NODE-VALUE( ) raises an error if any of the following occurs:

- The node is read only.
- The node contains invalid XML characters.
- The data in *memptr* is not null-terminated.
- *memptr* is not set to the exact size of the valid data.

For more information on accessing XML documents using the Document Object Model (DOM) interface, see *OpenEdge Development: Programming Interfaces*.

MENU-BAR attribute

The handle of a menu bar widget associated with a window.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: WINDOW widget

You can establish the menu bar for a window by assigning the MENU-BAR attribute.

MENU-KEY attribute

The accelerator key sequence that activates the pop-up menu for a widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

Any value you set must evaluate to a valid Progress key label, such as "a", "F1", or "ALT-SHIFT-F1".

MENU-MOUSE attribute (Graphical interfaces only)

The mouse button on a three-button mouse that activates the pop-up menu for a widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

Table 73 lists each mouse button and the attribute value that specifies it as the pop-up menu button.

Table 73: Pop-up menu button

Mouse button	Attribute value
Left	1
Middle	2
Right	3

If you use a two-button mouse, setting this attribute to “2" makes it impossible to access the menu with your mouse. If you do not set this attribute, it returns the Unknown value (?).

MERGE-BY-FIELD attribute

Specifies whether Progress merges changes on a field-by-field basis when saving changes from a ProDataSet temp-table buffer to the associated data source using the SAVE-ROW-CHANGES( ) method. The default value is TRUE.

Note: Merging a large number of changes from a ProDataSet object to the data source on a field-by-field basis is slower than saving changes buffer-by-buffer.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-source object handle

If TRUE, Progress performs a field-to-field comparison to identify data conflicts. In this case, a data conflict exists when the same field in the ProDataSet buffer and its associated data source buffer has changed. A data conflict would not exist if a different field had changed in either buffer.

If FALSE, Progress performs a buffer-to-buffer comparison to identify data conflicts. In this case, a data conflict exists when any field in the data source buffer has changed.

MERGE-CHANGES( ) method

Merges the changed rows from a ProDataSet object loaded with the GET-CHANGES( ) method into the corresponding rows in either a single temp-table or all temp-tables in the original ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle

Syntax

```
change-handle:MERGE-CHANGES( original-handle [, copy-all-mode ] )
```

change-handle

A handle to the ProDataSet object or ProDataSet temp-table buffer that contains the changed rows.

original-handle

A handle to the original ProDataSet object or ProDataSet temp-table buffer to merge with the changed rows.

copy-all-mode

An optional logical expression where TRUE indicates that Progress merge rows in a *copy-all* mode. When TRUE, Progress merges all after-image table rows whether or not they contain changes. In this case, the temp-table in the original ProDataSet object must have a unique primary index that Progress can use to find each corresponding row from the after-image table (since unchanged rows do not have a corresponding row in the before-image table). When a corresponding row is not found in the original ProDataSet object, Progress creates a new row using the row from the after-image table. When FALSE, Progress merges only after-image table rows that contain changes. The default value is FALSE.

For a ProDataSet object handle, all modified tables in the ProDataSet object are merged. For a Buffer object handle, only the temp-table associated with that buffer is merged.

If the ERROR attribute or REJECTED attribute for a changed table is TRUE, the MERGE-CHANGES( ) method backs out the changes. Otherwise, this method accepts the changes by copying the after-image table rows to the corresponding after-image table rows in the original ProDataSet temp-table. Progress also sets the BEFORE-ROWID attribute of the row in the after-image table to the Unknown value (?), sets the ROW-STATE of the row in the after-image table to ROW-UNMODIFIED (0), and removes the before-image table row (if it has one).

MERGE-ROW-CHANGES( ) method

Merges a single changed row from a ProDataSet object loaded with the GET-CHANGES( ) method into the corresponding row in the original ProDataSet temp-table buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
change-handle:MERGE-ROW-CHANGES( [ original-handle [, copy-all-mode ] ] )
```

change-handle

A handle to the ProDataSet before-image temp-table buffer for a changed row, or an after-image temp-table buffer for an unchanged row.

original-handle

A handle to the original ProDataSet temp-table buffer to merge with the changed row.

Progress uses *original-handle* only to match to the original table currently associated with *change-handle* (specified in the ORIGIN-HANDLE attribute). The current row in this table is ignored.

copy-all-mode

An optional logical expression where TRUE indicates that Progress merge the row in a *copy-all* mode. When TRUE, Progress merges the after-image table row whether or not it contains changes. In this case, the temp-table in the original ProDataSet object must have a unique primary index that Progress can use to find each corresponding row from the after-image table (since an unchanged row does not have a corresponding row in the before-image table). When a corresponding row is not found in the original ProDataSet object, Progress creates a new row using the row from the after-image table. When FALSE, Progress merges only an after-image table row that contain changes. The default value is FALSE.

If the ERROR attribute or REJECTED attribute for the changed row is TRUE, the MERGE-ROW-CHANGES( ) method backs out the change. Otherwise, this method accepts the change by copying the after-image table row to the corresponding after-image table row in the original ProDataSet temp-table. Progress also sets the BEFORE-ROWID attribute of the row in the after-image table to the Unknown value (?), sets the ROW-STATE of the row in the after-image table to ROW-UNMODIFIED (0), and removes the before-image table row (if it has one).

Note: The after-image table row, that contains the changes to the corresponding after-image table row in the original ProDataSet temp-table, contains changes from the original ProDataSet temp-table as well as any changes made in the associated data source row based on the MERGE-BY-FIELD attribute and PREFER-DATASET attribute settings in effect during the save operation.

MESSAGE-AREA attribute (Graphical interfaces only)

Controls the appearance of the message area in the window

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

You can set this attribute only before the window is realized.

MESSAGE-AREA-FONT attribute

The font number of the font used in the message area of a window.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

The font number represents an entry in the font table maintained by the FONT-TABLE handle.

MIN-BUTTON attribute (Windows only; Graphical interfaces only)

Determines whether the window has a minimize button in its caption bar.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

In Windows, a window can have maximize and minimize buttons depending on the settings of the MIN-BUTTON and MAX-BUTTON attributes. Both buttons are created on the window. If you set the MIN-BUTTON to TRUE and the MAX-BUTTON to FALSE, only the MIN-BUTTON is enabled; the MAX-BUTTON is disabled.

The MIN-BUTTON attribute must be set before the window is realized. The default value is TRUE.

On character platforms, this attribute has no effect.

MIN-COLUMN-WIDTH-CHARS attribute (Graphical interfaces only)

Sets the minimum width of a browse column in character units. If the browse:

- Has **not** been realized, all browse columns are minimally this size when realized.
- Has been realized, any browse column smaller than the specified minimum is increased to the minimum width.

Data type: Decimal

Access: Readable/Writeable

Applies to: BROWSE widget

The default value for the MIN-COLUMNS-WIDTH-CHARS attribute is equivalent to 1 pixel and depends on the display resolution and the size of the default font of the system.

An error occurs if you attempt to programmatically set the width of a browse column to a value smaller than the value specified with the MIN-COLUMN-WIDTH-CHARS attribute.

If COLUMN-RESIZABLE is set to TRUE, the user cannot change a column's width to be less than the minimum width specified with the MIN-COLUMN-WIDTH-CHARS attribute.

The MIN-COLUMN-WIDTH CHARS attribute affects the FIT-LAST-COLUMN attribute. Therefore, if you set FIT-LAST-COLUMN to TRUE, the last browse column is only resized to fit within the viewport if its width is no smaller than the minimum width. See the FIT-LAST-COLUMN attribute for more information about the FIT-LAST-COLUMN attribute.

When you assign a decimal value to an attribute representing a measurement in character units, Progress automatically rounds the assigned value to the nearest decimal value that corresponds to whole pixel units.

MIN-COLUMN-WIDTH-PIXELS attribute (Graphical interfaces only)

Sets the minimum width of a browse column in pixels. If the browse:

- Has **not** been realized, all browse columns are minimally this size when realized.
- Has been realized, any browse column smaller than the specified minimum is increased to the minimum width.

Data type: Integer

Access: Readable/Writeable

Applies to: BROWSE widget

The default minimum browse column width is 1 pixel.

An error occurs if you attempt to programmatically set the width of a browse column to a value smaller than the value specified using the MIN-COLUMN-WIDTH-PIXELS attribute.

If COLUMN-RESIZABLE is set to TRUE, the user cannot change a column's width to be less than the minimum width specified with the MIN-COLUMN-WIDTH-PIXELS attribute.

The MIN-COLUMN-WIDTH PIXELS attribute affects the FIT-LAST-COLUMN attribute. As a result, if FIT-LAST-COLUMN is set to TRUE, the last browse column is only resized to fit within the viewport if its width is no smaller than the minimum width. See the FIT-LAST-COLUMN attribute for more information about the FIT-LAST-COLUMN.

MIN-HEIGHT-CHARS attribute

The minimum height of a window, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: WINDOW widget

MIN-HEIGHT-PIXELS attribute

The minimum height of a window, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

MIN-SCHEMA-MARSHAL attribute

Set to TRUE to minimize schema information when marshaling data for a temp-table parameter. The temp-table may be an independent temp-table or a member of a ProDataSet object.

This attribute is supported only for backward compatibility. Use the SCHEMA-MARSHAL attribute instead.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Temp-table object handle

This attribute suppresses index descriptions and some field information (such as label, help, field validation expression, and so on) when marshaling data. It does marshal field names, data types, and extents.

The MIN-SCHEMA-MARSHAL attribute corresponds to the SCHEMA-MARSHAL attribute with a value of "MIN".

Note: If you specify both the MIN-SCHEMA-MARSHAL attribute and the SCHEMA-MARSHAL or NO-SCHEMA-MARSHAL attribute for an individual temp-table, Progress uses the attribute you most recently specified.

Setting this attribute overrides the setting of the Temp-table Schema Marshal (`-ttmarshal`) startup parameter for an individual temp-table parameter. For more information about this startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

See also NO-SCHEMA-MARSHAL attribute, SCHEMA-MARSHAL attribute

MIN-VALUE attribute

The minimum value of a slider.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SLIDER widget

You can set this attribute only before the widget is realized.

MIN-WIDTH-CHARS attribute

The minimum width of a window, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: WINDOW widget

MIN-WIDTH-PIXELS attribute

The minimum width of a window, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

MODIFIED attribute

Indicates whether the value of the SCREEN-VALUE attribute for the widget has changed.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

For browse columns, this attribute is readable only.

For all widgets, the MODIFIED attribute is set to TRUE when the SCREEN-VALUE attribute for the widget is changed, whether or not the field is enabled. For all widgets **except** the editor widget, the MODIFIED attribute is set to TRUE if the SCREEN-VALUE attribute for the widget is changed using a 4GL statement, such as assignment or DISPLAY. DISPLAY sets MODIFIED to TRUE only when the field is enabled. You can then reset the attribute to FALSE for each widget that can receive input focus or otherwise change value after it is initially displayed.

For editors, the successful execution of either the SAVE-FILE( ) or the READ-FILE( ) methods sets the MODIFIED attribute to FALSE.

For browses, if any browse cell changes, Progress sets MODIFIED to TRUE. The application can reset MODIFIED to FALSE as necessary. If the query associated with a browse is reopened, Progress resets MODIFIED to FALSE.

If the widget is not already realized and you reference its MODIFIED attribute, Progress realizes the widget.

MOUSE-POINTER attribute

Returns the name of the mouse pointer loaded by LOAD-MOUSE-POINTER( ).

Data type: CHARACTER

Access: Readable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

MOVABLE attribute (Graphical interfaces only)

Indicates whether the widget can receive direct manipulation events.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Set MOVABLE to TRUE to enable users to move the widget. To enable users to move more than one widget at a time, you must also set the SELECTABLE attribute to TRUE for each widget.

Note: Setting the MOVABLE attribute to TRUE enables direct manipulation events for the widget. These events take precedence over all other events. This effectively prevents data entry using the widget until all direct manipulation events are disabled (that is, until MOVABLE, RESIZABLE, and SELECTABLE are all FALSE).

MOVE-AFTER-TAB-ITEM( ) method

Assigns the method widget to the tab position after a specified widget. Both the method widget and the specified widget must be in the same field group.

Return type: LOGICAL

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

Syntax

```
MOVE-AFTER [ -TAB-ITEM ] ( widget-handle )
```

widget-handle

A handle to the widget after whose tab position you want to move the method widget.

If the operation is successful, the method returns TRUE. To set the first or last tab position, set the FIRST-TAB-ITEM or LAST-TAB-ITEM attribute (respectively) for the field group.

If *widget-handle* specifies a frame, the tab order of the method widget is positioned so that it follows the last widget parented by the frame in that frames own tab order. For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry in this manual and the chapter on frames in *OpenEdge Development: Progress 4GL Handbook*.

Note: Any tab reordering that you do with this method can be reset by a subsequent ENABLE statement unless you define the frame that owns the field group with the KEEP-TAB-ORDER option. For more information, see the ENABLE statement and Frame phrase reference entries.

MOVE-BEFORE-TAB-ITEM( ) method

Assigns the method widget to the tab position before a specified widget. Both the method widget and the specified widget must be in the same field group.

Return type: LOGICAL

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

Syntax

```
MOVE-BEFORE [ -TAB-ITEM ] ( widget-handle )
```

widget-handle

A handle to the widget before whose tab position you want to move the method widget.

If the operation is successful, the method returns TRUE. To set the first or last tab position, set the FIRST-TAB-ITEM or LAST-TAB-ITEM attribute (respectively) for the field group.

If *widget-handle* specifies a frame, the tab order of the method widget is positioned so that it precedes the first widget parented by the frame in that frames own tab order. For more information on how frames owned by a field group participate in the tab order of that field group, see the FRAME widget reference entry in this manual and the chapter on frames in *OpenEdge Development: Progress 4GL Handbook*.

Note: Any tab reordering that you do with this method can be reset by a subsequent ENABLE statement unless you define the frame that owns the field group with the KEEP-TAB-ORDER option. For more information, see the ENABLE statement and Frame phrase reference entries.

MOVE-COLUMN( ) method (Graphical interfaces only)

Repositions a column in a browse widget.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
MOVE-COLUMN ( source , destination )
```

source

An integer expression specifying the column to be moved.

destination

An integer expression specifying the position to which the column is moved.

The columns of a browse are numbered left to right beginning with 1 including both visible and hidden columns. For example, *browse*:MOVE-COLUMN(1, 3) moves the first column to the third position (the second column becomes the first column and the third column becomes the second column). If the column is successfully moved, the method returns the value TRUE.

MOVE-TO-BOTTOM( ) method

Moves the widget to the bottom (or back) of other widgets of the same class on the display.

Return type: LOGICAL

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
MOVE-TO-BOTTOM ( )
```

For the purposes of this method, the classes are as follows:

- Windows
- Frames
- Images and rectangles
- All other field-level widgets

If the operation is successful, the method returns TRUE.

When you use this method, set the KEEP-FRAME-Z-ORDER attribute to TRUE.

In character interfaces, the MOVE-TO-BOTTOM method applies only to the Frame.

MOVE-TO-EOF( ) method

Moves the cursor position in an editor to the end of the current text.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
MOVE-TO-EOF ( )
```

If the operation is successful, the method returns TRUE.

MOVE-TO-TOP( ) method

Moves the widget to the top (or front) of other widgets of the same class on the display.

Return type: LOGICAL

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

Syntax

```
MOVE-TO-TOP ( )
```

For the purposes of this method, the classes are as follows:

- Windows
- Frames
- Images and rectangles
- All other field-level widgets

If the operation is successful, the method returns TRUE.

Images and rectangles are displayed behind other field-level widgets and cannot be moved on top of them.

When you use this method, set the KEEP-FRAME-Z-ORDER attribute to TRUE.

Note: In character interfaces, the MOVE-TO-TOP method applies only to the Frame.

MULTI-COMPILE attribute

Specifies whether Progress compiles all class definition files in the inherited class hierarchy or only those class definition files for which a cached version is not found.

Note: This attribute is applicable only when compiling class definition (`.cls`) files.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: COMPILER system handle

When set to TRUE, Progress compiles only those class definition files in the inherited class hierarchy that are not found in the cache. Progress also caches any classes or interfaces it compiles to avoid recompiling them during the session.

When set to FALSE, Progress compiles all class definition files in the inherited class hierarchy. Progress also clears the cache of any classes or interfaces compiled during the session. The default value is FALSE.

MULTIPLE attribute

Indicates the selection behavior of browse selection list widgets, and the read and write behavior of the system clipboard.

Data type: LOGICAL

Access: Readable/Writeable (Read-only for browse)

Applies to: BROWSE widget, CLIPBOARD system handle, SELECTION-LIST widget

For browse widgets, the MULTIPLE attribute specifies whether the user can select multiple rows from the widget, or only a single row. (Typically, the selected rows are processed in response to a DEFAULT-ACTION event.) The MULTIPLE attribute is read-only for browse widgets. The MULTIPLE attribute for a browse can be set before the browse is realized. Use the MULTIPLE or SINGLE option of the browse phrase in the DEFINE BROWSE statement to set the selection behavior for a browse widget. The MOUSE-SELECT-DOWN and MOUSE-SELECT-UP events are generated as the use scrolls through the browse.

Note: When an updateable browse is in edit mode, a cell has focus, all other selected rows are deselected.

For selection-list widgets, the MULTIPLE attribute specifies whether the user can select multiple items from the widget, or only a single item. (Typically, the selected rows are processed in response to a DEFAULT-ACTION event.) You can specify selection behavior for a selection list using the MULTIPLE or SINGLE option of a SELECTION-LIST phrase. You can set this attribute for a selection list only before the widget is realized.

Note: When a selection-list has the MULTIPLE attribute, the selection of an item does not clear any of the items previously selected. An item remains highlighted until the selection-list is cleared.

For the CLIPBOARD handle, the MULTIPLE attribute specifies whether Progress reads and writes data to the CLIPBOARD handle as multiple items or as a single item. For more information, see the CLIPBOARD system handle reference entry.

MULTITASKING-INTERVAL attribute (Windows only)

How often Progress filters events between itself and other Windows applications.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

The value of the MULTITASKING-INTERVAL attribute determines how often Progress internally filters events (messages) between itself and other Windows applications. As Progress filters these events more often, it executes procedures less efficiently, but allows other windows applications more opportunity to execute. Adjusting the internal event filter is particularly useful during background processing, such as report generation.

The default value, zero, tells Progress never to filter events internally, giving OpenEdge applications maximum access to execution resources. This is perfectly adequate for interactive OpenEdge applications that block for input often, giving other applications enough opportunity to execute.

For values greater than zero, the lower the value, the more often Progress internally filters events, giving other applications greater opportunity to execute, but slowing down Progress execution. However, similar to a TRUE value for the IMMEDIATE-DISPLAY attribute, low non-zero values also cause Progress to refresh the display more often, potentially providing crisper display interaction. Low non-zero values also provide better interoperability with other applications, for example, using Dynamic Data Exchange (DDE).

The maximum value you can set is 9999. In general, set this attribute greater than zero only for code segments that perform lengthy background operations, and reset it to zero before the application blocks for interactive input (for example, executes a WAIT-FOR or UPDATE statement). This attribute provides the same functionality as the MultitaskingInterval parameter in the current environment, which might be the Registry (Windows only) or an initialization file. For more information on environments, see the chapter on user interface environments in *OpenEdge Deployment: Managing 4GL Applications*.

MUST-UNDERSTAND attribute

Indicates whether a SOAP-header-entryref object is mandatory (TRUE) or optional (FALSE) for the recipient to process.

Data type: LOGICAL

Access: Readable

Applies to: SOAP-header-entryref object handle

If the SOAP-header-entryref object does not contain a MUST-UNDERSTAND attribute, this attribute returns FALSE.

Name property (Windows only; Graphical interfaces only)

The name of the control-frame and control-frame COM object.

Return type: CHARACTER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the NAME attribute of the corresponding control-frame widget to the same value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.

Caution: If you change the value of this property at run time, any OCX event procedures that you have defined for a corresponding ActiveX control will not respond to control events because the events are sent with the new name.

NAME attribute

A string identifier for the specified object or widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Asynchronous request object handle, BROWSE widget (browse and cell), Buffer object handle, Buffer-field object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, ProDataSet object handle, Data-relation object handle, Data-source object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, Query object handle, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, Server object handle, Server socket object handle, SLIDER widget, SOAP-header object handle, SOAP-header-entryref object handle, Socket object handle, SUB-MENU widget, Temp-table object handle, TEXT widget, THIS-PROCEDURE system handle, TOGGLE-BOX widget, WINDOW widget, X-document object handle, X-noderef object handle

For the SOAP-header and SOAP-header-entryref object handles, this attribute is read-only.

For static data representation widgets, the default value is the name of the field or variable associated with the widget. You can use the NAME attribute to store any information associated with the widget.

For a state-reset, state-aware, or stateless AppServer, this attribute returns the connection ID for the AppServer associated with the server handle. For a state-free AppServer, it returns the connection ID of the first AppServer connection created in the session pool with this server handle when the application service is first bound using the CONNECT( ) method. The default value is the unique connection name of the AppServer.

For Web services, the default value is the URL of the Web service procedure object from the WSDL. This is the portType name used on the RUN statement that instantiated this object.

For a non-Web service procedure, returns the pathname of the procedure file that contains the current procedure.

For control-frames, this attribute maps to the Name property of the of the control-frame COM object (ActiveX control container).

For dynamic widgets and asynchronous request handles, this attribute defaults to the Unknown value (?).

Caution: If you change the value of this property at run time, any OCX event procedures that you have defined for a corresponding ActiveX control will not respond to control events because the events are sent with the new name.

For query objects, the NAME attribute applies only to static queries. For more information on query objects, see *OpenEdge Development: Progress 4GL Handbook*.

For the SOAP-header object handle, this attribute is the qualified name of the SOAP-header object, which consists of a namespace prefix + ":" + HEADER.

For the SOAP-header-entryref object handle, this attribute is the qualified name of the SOAP-header-entryref object, which consists of a namespace prefix + ":" + localname. You cannot change this attribute directly; you must use the local-name and namespace-prefix.

For temp-table objects, this attribute is read-only and returns the name of the temp-table as specified in the TEMP-TABLE-PREPARE method. If TEMP-TABLE-PREPARE( ) has not been called or the name has been cleared by CLEAR( ) and no subsequent TEMP-TABLE-PREPARE( ) has been called, then this attribute returns the Unknown value (?), which means that the table is in the UNPREPARED state.

For the X-document object handle or X-noderef object handle, this attribute returns the name of the XML node.

For any object or widget, this attribute can contain any arbitrary value that you set.

NAMESPACE-PREFIX attribute

This attribute returns or sets the qualified part of a namespace-aware XML node name (that is, the prefix before the colon character). The prefix is used to identify elements that belong to the namespace associated with the prefix (as set by the NAMESPACE-URI attribute). For nodes created with the CREATE-NODE( ) method, or nodes of any type other than ELEMENT or ATTRIBUTE, this attribute returns the Unknown value (?).

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle, X-document object handle, X-noderef object handle

This attribute is read-only for the X-document object handle.

NAMESPACE-URI attribute

The namespace URI of a namespace-aware XML node name, a SOAP-header-entryref object, or a ProDataSet or Temp-Table element and its child elements. The namespace of an XML document is used to scope XML attributes and elements. For nodes created with the CREATE-NODE( ) method, or nodes of any type other than ELEMENT or ATTRIBUTE, this attribute returns the Unknown value (?).

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer object handle, ProDataSet object handle, SOAP-header-entryref object handle, Temp-table object handle, X-document object handle, X-noderef object handle

This attribute is read-only for the SOAP-header-entryref object handle, X-document object handle, and X-noderef object handle.

NEEDS-APPSERVER-PROMPT attribute

Indicates whether WebClient should prompt for AppServer connection parameters, if it does not find those values in the security cache, (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Readable

Applies to: CODEBASE-LOCATOR system handle

Valid only if LOCATOR-TYPE is "AppServer".

NEEDS-PROMPT attribute

Indicates whether WebClient should prompt for an Internet server userid and password, if it does not find those values in the security cache, (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Readable

Applies to: CODEBASE-LOCATOR system handle

NESTED attribute

Indicates whether child rows of a ProDataSet temp-table buffer are nested within their parent rows when writing the XML representation of a ProDataSet object that contains data-relations. This also causes the XML Schema definitions for the related temp-tables to be nested.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-relation object handle

See also ADD-RELATION( ) method, DEFINE DATASET statement, WRITE-XML( ) method, WRITE-XMLSCHEMA( ) method

NEW attribute

Indicates whether the record in the buffer is newly created. If the record is newly created, NEW is TRUE. If the record in the buffer was read from the database, NEW is FALSE.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle

Note: The NEW attribute corresponds to the NEW function.

NEW-ROW attribute

Indicates whether the focused browse row exists in the database.

Data type: LOGICAL

Access: Readable

Applies to: BROWSE widget

If this attribute is set to TRUE, the row in focus was added to the browse using the INSERT-ROW( ) method and has not been added to the database.

NEXT-COLUMN attribute

The widget handle of the next sibling, in physical order, of the current browse column whether or not the column is visible. The browse MOVE-COLUMN method changes the physical order of columns and updates this attribute accordingly.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: BROWSE widget (column)

NEXT-ROWID attribute

Provides the ROWID of the data source row at which the next FILL operation should start. Progress sets this attribute after each FILL operation in a series of FILL operations to retrieve data source rows in batches. You typically assign the value of this attribute to the RESTART-ROWID attribute before each FILL operation.

Note: This attribute is not marshalled between the client and the AppServer. You are responsible for retrieving, storing, and transporting this attribute value between the client and the AppServer.

Data type: ROWID

Access: Readable/Writeable

Applies to: Data-source object handle

Syntax

```
data-source-handle:NEXT-ROWID( buffer-sequence-number | buffer-name )
```

data-source-handle

The handle to the Data-source object.

buffer-sequence-number

An INTEGER that represents the sequence number of a buffer in the list of buffers for the Data-source object. Specify *buffer-sequence-number* to identify a buffer in the Data-source object when the Data-source object is defined against more than one database table buffer. The default is the first (or only) buffer in the Data-source object.

Note: Sequence numbers for buffers in a Data-source object start at one, where one represents the top level and subsequent numbers represent lower levels of join, if any.

buffer-name

A CHARACTER expression that evaluates to the name of a buffer in the list of buffers for the Data-source object.

If an invalid buffer is specified, this attribute returns the Unknown value (?).

It is best to use the NEXT-ROWID attribute with a top-level ProDataSet temp-table, or a child temp-table that has only one parent record, because Progress sets this attribute on the child temp-table for each parent record (as opposed to once per child temp-table).

Use this attribute when retrieving batches of data source rows containing stable data. Otherwise, it might be better to use a unique index to reopen the query associated with the Data-source object to retrieve a specific batch of data source rows.

See also BATCH-SIZE attribute, FILL( ) method, RESTART-ROWID attribute

NEXT-SIBLING attribute

The next entry in a list of handles, relative to a given handle.

Data type: HANDLE

Access: Readable

Applies to: Asynchronous request object handle, BROWSE widget, Buffer object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, ProDataSet object handle, Data-source object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, Server object handle, SLIDER widget, Socket object handle, Server socket object handle, SESSION system handle, SUB-MENU widget, TEXT widget, THIS-PROCEDURE system handle (and all procedure handles), TOGGLE-BOX widget, WINDOW widget.

Table 74 summarizes the value of NEXT-SIBLING for each relevant handle type.

Table 74: NEXT-SIBLING attribute values by handle type *(1 of 2)*

Handle type	Value of NEXT-SIBLING
Asynchronous Request	The handle of the next asynchronous request submitted for execution on the AppServer or Web Server that is running the specified request.
Procedure	The handle of the next persistent procedure in the current OpenEdge session. If the current procedure is a proxy for a persistent procedure running on an AppServer or for a Web service, specifies the next procedure object bound to the same server handle.
Server	The next server handle created in the current OpenEdge session (independent of subtype).
Socket and Server-socket	The next socket handle in the chain of socket handles for the current OpenEdge session. Returns the Unknown value (?) for the last handle in the chain.
ProDataSet object	The handle to the next dynamic ProDataSet object in the chain of ProDataSet objects for the current OpenEdge session, which is available after using the SESSION:FIRST-DATASET attribute.
Dynamic query	The handle to the next dynamic query in the chain of dynamic queries for the current OpenEdge session, which is available after using the SESSION:FIRST-QUERY attribute.
Dynamic Data-source object	The handle to the next dynamic Data-source object in the chain of dynamic Data-source objects for the current OpenEdge session, which is available after using the SESSION:FIRST-DATA-SOURCE attribute.

Table 74: NEXT-SIBLING attribute values by handle type *(2 of 2)*

Handle type	Value of NEXT-SIBLING
Buffer object	The handle to the next buffer object in the session buffer list, which is available after using the SESSION:FIRST-BUFFER attribute.
Widget	The handle of the next widget in the widget list. **Note:** A widget must first be realized before it can become part of the list. A hidden widget cannot become part of the list since it is not realized. A widget that is already part of the list can be hidden and it remains part of the list.

If the given handle is the last handle in the list, NEXT-SIBLING assumes the value of an invalid handle. To check the validity of a handle, use the VALID-HANDLE function.

See also PREV-SIBLING attribute

NEXT-TAB-ITEM attribute

The widget handle of the next widget in the tab order of a field group relative to the specified widget.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

The NEXT-TAB-ITEM attribute returns the Unknown value (?) for a widget that is at the end of the tab order in a field group.

NO-CURRENT-VALUE attribute

The default behavior for a slider is to display the current value for a given position on a slider control. The NO-CURRENT-VALUE attribute allows you to override this default behavior.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SLIDER widget

NO-EMPTY-SPACE attribute (Graphical interface only)

Allows the browse to display with no empty space to the right and no horizontal scroll bar.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

When the last browse column can be fully displayed in the browse viewport with empty space to the right, you can use NO-EMPTY-SPACE attribute to widen the width of the last browse column so that the column fits within the viewport with no empty space to its right and no horizontal scroll bar.

The default value is FALSE.

The following shows the DEFINE BROWSE statement syntax with NO-EMPTY-SPACE specified:

```
DEFINE BROWSE b1 QUERY q1
DISPLAY customer.cust-num customer.name
ENABLE customer.cust-num WITH  3 DOWN WIDTH 40 NO-EMPTY-SPACE
```

NO-EMPTY-SPACE is primarily intended for use in the initial layout of a static browse. It is most useful when laying out a browse with a specified width when you have only a few browse columns, and you want to fully use the available space in your viewport.

If the NO-EMPTY-SPACE is set to TRUE and there is empty space, the last browse column is widened to fill up the space. Also, if any browse column's width attribute is changed or the browse's width attribute is changed so that the last browse column is fully displayed in the browse's viewport with empty space to its right, then the last browse column's width is widened so that it fits within the viewport with no empty space and no horizontal scroll bar.

NO-EMPTY-SPACE never reduces the width of the last browse column.

NO-EMPTY-SPACE is ignored under the following circumstances:

- When the last browse column's width is explicitly set at run time after the browse is realized.
- When the last browse column displays partially or entirely outside of the viewport.
- If NO-EMPTY-SPACE is set to FALSE, the last browse column's width remains the same and is never changed by Progress.
- If you specify NO-EMPTY-SPACE for an individual browse, and the `-expandbrow` startup parameter is also specified, then the NO-EMPTY-SPACE attribute overrides `-expandbrow` for that browse.
- When you use NO-EMPTY-SPACE, the original width of the last browse column is not remembered. For example, if the original width of the last browse column is 48 pixels, the DEFINE BROWSE statement has NO-EMPTY-SPACE specified, and if at run time there are 12 blank pixels as empty space in the right side of the viewport, the last browse column's width is increased to 60 pixels, so there is no blank space in the browse.
- If later at run time, the width of a column other than the last column is increased, a horizontal scroll bar is added to the browse. The width for the last browse column remains at 60 pixels.

NO-FOCUS attribute (Windows only)

Determines whether a button can accept focus. A button for which the NO-FOCUS attribute is TRUE will not take focus when the mouse is clicked on it and it will not accept keyboard input. Also, Progress will not generate ENTRY or LEAVE events for the button. NO-FOCUS buttons behave similarly to standard Windows toolbar buttons.

This attribute must be set before the button is realized. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BUTTON widget

A button for which the NO-FOCUS attribute is TRUE will not be added to its parent frame's tab order. If the NO-FOCUS attribute is switched from TRUE to FALSE before the button is realized, the button will be added to the end of its parent frame's tab order. Switching the NO-FOCUS option from FALSE to TRUE before realization will remove the button from its parent frame's tab order.

The mnemonic key (**ALT** accelerator) for a widget will not work if the widget is removed from the tab order. Also, because the widget is not in the tab order, pressing **TAB** will not change focus from the widget.

Keep in mind that if a frame that contains a NO-FOCUS button does not itself have focus, the frame will not receive focus when the button is pushed. In this situation, frame entry or leave events are not generated. Focus stays on the current widget when a NO-FOCUS button is pushed, even across multiple frames in a window.

NONAMESPACE-SCHEMA-LOCATION attribute

Determines the location the XML Schema file to validate when elements do not contain a namespace.

Applies to: CHARACTER

Access: Readable/Writable

Applies to: X-document object handle, SAX-reader object handle

Contains the XML Schema file location for elements with no namespace.

This attribute specifies a single schema location. It defaults to an empty string ("").

NO-SCHEMA-MARSHAL attribute

Set to TRUE to exclude schema information when marshaling data for a temp-table parameter. The temp-table may be an independent temp-table or a member of a ProDataSet object.

This attribute is supported only for backward compatibility. Use the SCHEMA-MARSHAL attribute instead.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Temp-table object handle

The receiving procedure must have a temp-table definition in which to receive the temp-table. If the receiving temp-table is dynamic, and it does not have a definition, Progress generates a run-time error.

Where this attribute suppresses index descriptions and all field information when marshaling data, Progress cannot perform field validation. Be sure the schema of both the source and target temp-tables is the same.

The NO-SCHEMA-MARSHAL attribute corresponds to the SCHEMA-MARSHAL attribute with a value of "NONE".

Note: If you specify both the NO-SCHEMA-MARSHAL attribute and the SCHEMA-MARSHAL or MIN-SCHEMA-MARSHAL attribute for an individual temp-table, Progress uses the attribute you most recently specified.

Setting this attribute overrides the setting of the Temp-table Schema Marshal (`-ttmarshal`) startup parameter for an individual temp-table parameter. For more information about this startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

See also MIN-SCHEMA-MARSHAL attribute, SCHEMA-MARSHAL attribute

NO-VALIDATE attribute

Specifies that Progress ignore the validation conditions in the schema for all fields in a dynamic browse.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

This attribute is readable by both static and dynamic browses, and writeable only by dynamic browses.

If TRUE, Progress does not run the validation for dynamic browse columns. If FALSE, Progress runs the validation. The default value is FALSE.

Note: For a static browse, this attribute is set in the DEFINE BROWSE statement using the NO-VALIDATE option.

NODE-VALUE attribute

Returns (or sets) the value of the XML node.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: X-noderef object handle

The following example demonstrates the use of the NODE-VALUE attribute:

```
IF hNoderef:NODE-VALUE = "500"
   THEN hNoderef:NODE-VALUE = "1000".
```

NODE-VALUE-TO-LONGCHAR( ) method

Copies the contents of an XML X-noderef node to a LONGCHAR, and optionally converts the contents to a specific code page.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
NODE-VALUE-TO-LONGCHAR( longchar [ codepage ] )
```

longchar

An expression of type LONGCHAR.

codepage

A character-string expression that evaluates to the name of a code page. The name you specify must be a valid code page name available in the DLC/convmap.cp file. If you do not specify *codepage* and the code page of *longchar* is fixed (that is, set using the FIX-CODEPAGE function), Progress converts *longchar* to the fixed code page. If you do not specify *codepage* and the code page of *longchar* is not fixed, Progress converts *longchar* to the code page specified by -cpinternal. If you specify *codepage* and the code page of *longchar* is fixed, they must agree. Otherwise, Progress raises a run-time error.

NODE-VALUE-TO-LONGCHAR( ) frees the memory currently allocated by *longchar* (if any), allocates sufficient memory to the LONGCHAR to accommodate the node, and copies the node to the LONGCHAR.

If X-NODEREF:NODE-VALUE is the empty string (""), the resulting *longchar* has a size of zero length.

For more information on accessing XML documents using the Document Object Model (DOM) interface, see *OpenEdge Development: Programming Interfaces*.

NODE-VALUE-TO-MEMPTR( ) method

Copies the contents of an XML X-noderef node to a MEMPTR. This makes it easier to manipulate when its length exceeds the Progress limit for text strings, which is approximately 32K.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
NODE-VALUE-TO-MEMPTR( memptr )
```

memptr

An expression of type MEMPTR.

NODE-VALUE-TO-MEMPTR( ) frees the memory currently allocated by *memptr* (if any), allocates sufficient memory to the MEMPTR to accommodate the node, and copies the node to the MEMPTR.

If X-NODEREF:NODE-VALUE is “” (the empty string), the resulting *memptr* has a size of zero.

Note: When you no longer need the memory used by *memptr*, you must free it yourself. To do so, use the SET-SIZE statement.

The following fragment uses NODE-VALUE-TO-MEMPTR( ) to access a large text node in chunks:

```
IF my-xnoderef:SUBTYPE = "#TEXT" THEN DO:
  IF my-xnoderef:LENGTH > 32000 THEN DO:
    success = my-xnoderef:NODE-VALUE-TO-MEMPTR(my-memptr).
    pos = 1.
    len = 32000.
    DO WHILE pos < GET-SIZE(my-memptr):
      my-nodeval = GET-STRING(my-memptr, pos, len).
      /* do something with my-nodeval */
      pos = pos + len.
    END.
    SET-SIZE(my-memptr) = 0.
  END.
  ELSE
    my-nodeval = my-xnoderef:NODE-VALUE.
END.
```

For more information on accessing XML documents using the Document Object Model (DOM) interface, see *OpenEdge Development: Programming Interfaces*.

NORMALIZE( ) method

Normalizes TEXT and ATTRIBUTE nodes in the full depth of the sub-tree under this XML node.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
NORMALIZE ( )
```

The NORMALIZE( ) method normalizes TEXT nodes by removing empty TEXT nodes and merging adjacent TEXT nodes. Thus, only structure node types (such as ELEMENT, CDATA-SECTION, and so on) separate TEXT nodes. The NORMALIZE( ) method also normalizes white space in ATTRIBUTE nodes according to the rules defined by the XML specification.

NUM-BUFFERS attribute

The number of buffers in a query or ProDataSet object.

Data type: INTEGER

Access: Readable

Applies to: ProDataSet object handle, Query object handle

NUM-BUTTONS attribute

The number of items in a radio set.

Data type: INTEGER

Access: Readable

Applies to: RADIO-SET widget

NUM-CHILD-RELATIONS attribute

The number of relations for which the buffer is the parent. A buffer may be a parent in multiple relations, but a child in only one.

Data type: INTEGER

Access: Readable

Applies to: Buffer object handle

NUM-CHILDREN attribute

Returns the number of child nodes below the node referred to by a node reference. Attributes are not counted since they are not considered children of a node.

Data type: INTEGER

Access: Readable

Applies to: X-document object handle, X-noderef object handle

The following example demonstrates getting all the child nodes from the XML node referenced by hNoderef using the NUM-CHILDREN attribute:

```
REPEAT j = 1 TO hNoderef:NUM-CHILDREN:
 hNoderef:GET-CHILD(hNoderefChild,j).
   . . .
END.
```

NUM-COLUMNS attribute

The number of columns in a browse. This number includes hidden as well as visible columns.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget

NUM-DROPPED-FILES attribute (Windows only; Graphical interfaces only)

Indicates the number of files dropped in the last drag-and-drop operation performed on the widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

If there is no current drag-and-drop operation, NUM-DROPPED-FILES returns the Unknown value (?).

NUM-ENTRIES attribute (Graphical interfaces only)

The number of entries in a color table or font table.

Data type: INTEGER

Access: Readable/Writeable

Applies to: COLOR-TABLE system handle, FONT-TABLE system handle

This attribute returns zero (0) in character interfaces because colors and fonts are not supported for character interfaces.

NUM-FIELDS attribute

The number of fields defined in the buffer's table.

Data type: INTEGER

Access: Readable

Applies to: Buffer object handle

NUM-FORMATS attribute

The number of formats available for reading the data currently stored in the clipboard.

Data type: INTEGER

Access: Readable

Applies to: CLIPBOARD system handle

If there are no formats available, the attribute returns 0. For more information, see the reference entry for the CLIPBOARD system handle.

NUM-HEADER-ENTRIES attribute

The number of SOAP-header-entryref object entries attached to the SOAP-header object.

Data type: INTEGER

Access: Readable

Applies to: SOAP-header object handle

NUM-ITEMS attribute

The number of entries in a combo box, SAX-attributes object, or selection list.

Data type: INTEGER

Access: Readable

Applies to: COMBO-BOX widget, SAX-attributes object handle, SELECTION-LIST widget

NUM-ITERATIONS attribute

The number of currently visible foreground iterations for a frame or the number of rows currently visible in a browse widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget, FRAME widget

NUM-LINES attribute

The number of lines in an editor widget.

Data type: INTEGER

Access: Readable

Applies to: EDITOR widget

Lines are substring of the editor field or variable that are terminated by end-of-line characters. The editor inserts end-of-line characters at the current cursor position it receives a RETURN event.

NUM-LOCKED-COLUMNS attribute

The number of visible leading columns locked in a browse widget. If a locked column is hidden, the next visible non-locked column in the browse will then become locked.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget

When you use the horizontal scrollbar to scroll columns in the browse, locked columns do not move. For example, if NUM-LOCKED-COLUMNS is 3, then the three leftmost columns in the browse are locked.

Note: In character mode, this attribute can only be set before the widget is realized.

NUM-LOG-FILES attribute

The number of rolled over log files to keep on disk at any one time, for OpenEdge session, including the current log file.

Data type: INTEGER

Access: Readable

Applies to: LOG-MANAGER system handle

Valid values are:

- **0** — This means there is no limit on the number of log files to keep.
- **2 or greater** — The default is 3.

The NUM-LOG-FILES attribute corresponds to the Number of Log Files to Keep (`-numlogfiles`) startup parameter.

Use the LOG-THRESHOLD attribute or the Log Threshold (`-logthreshold`) startup parameter to specify the file size at which OpenEdge rolls over (renames and saves) log files.

For more information about the Number of Log Files to Keep (`-numlogfiles`) and Log Threshold (`-logthreshold`) startup parameters, see *OpenEdge Deployment: Startup Command and Parameter Reference* and *OpenEdge Development: Debugging and Troubleshooting*.

NUM-MESSAGES attribute

The number of error messages currently available through the ERROR-STATUS handle.

Data type: INTEGER

Access: Readable

Applies to: ERROR-STATUS system handle

NUM-PARAMETERS attribute

The number of parameters expected.

Data type: INTEGER

Access: Readable/Writable

Applies to: CALL object handle

Syntax

```
NUM-PARAMETERS(integer-expression)
```

integer-expression

An INTEGER expression indicating the number of parameters expected. The default is zero.

Whenever NUM-PARAMETERS is set, all existing parameters, including those from earlier uses of SET-PARAMETER, are cleared and deallocated.

If there are parameters to be passed, NUM-PARAMETERS must be set before the INVOKE( ) method is executed.

When you are getting a parameter, set NUM-PARAMETERS to the actual number of parameters to be passed, which might vary if one or more trailing parameters are optional.

When you are setting an attribute, use NUM-PARAMETERS 1.

NUM-REFERENCES attribute

The number of references to a Buffer, ProDataSet, or Temp-table object that is defined as a parameter to which reference-only objects are bound.

Data type: INTEGER

Access: Readable

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Use this attribute to determine whether a Buffer, ProDataSet, or Temp-table object is referenced by another procedure before you delete the defining procedure or the referenced object itself (if it is a dynamic object).

If the Buffer, ProDataSet, or Temp-table object is not referenced by any other procedure (or other such object), this attribute returns 0. Otherwise, this attribute returns the number of procedures (or other such objects) currently referencing the object.

This attribute applies to objects defined as reference-only parameters, not shared objects.

See also DEFINE DATASET statement (REFERENCE-ONLY option), DEFINE TEMP-TABLE statement (REFERENCE-ONLY option), DELETE OBJECT statement, DELETE PROCEDURE statement, RUN statement

NUM-RELATIONS attribute

The number of data-relation objects in a ProDataSet object.

Data type: INTEGER

Access: Readable

Applies to: ProDataSet object handle

NUM-REPLACED attribute

Indicates the number of occurrences replaced by the last REPLACE( ) method executed for the Editor.

Data type: INTEGER

Access: Readable

Applies to: EDITOR widget

If the Editor has not yet been realized, the attribute has the Unknown value (?).

NUM-RESULTS attribute

The number of rows currently in a query's result list.

Data type: INTEGER

Access: Readable

Applies to: Query object handle

Note: The NUM-RESULTS attribute corresponds to the NUM-RESULTS function.

See also NUM-RESULTS function

NUM-SELECTED-ROWS attribute

The number of rows currently selected in a browse widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget

A browse can have more than one row selected only if the MULTIPLE attribute is TRUE.

NUM-SELECTED-WIDGETS attribute

The number of top-level widgets in a frame or window that the user has selected for direct manipulation.

Data type: INTEGER

Access: Readable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a window, this attribute returns the number of selected frames. For a frame or dialog box, this attribute returns the number of selected field-level widgets. You can use the GET-SELECTED-WIDGET( ) method to access the individual selected widgets.

NUM-SOURCE-BUFFERS attribute

The number of source buffers in the Data-source object.

Data type: INTEGER

Access: Readable

Applies to: Data-source object handle

NUM-TABS attribute

The number of widgets in the field group with tab positions.

Data type: INTEGER

Access: Readable

Applies to: FIELD-GROUP widget

NUM-TO-RETAIN attribute

The number of frame iterations to retain when a down frame scrolls to a new set of iterations.

Data type: INTEGER

Access: Readable

Applies to: FRAME widget

This value is set using the RETAIN option of the Frame phrase.

NUM-TOP-BUFFERS attribute

The number of top-level buffers in a ProDataSet object.

Note: A top-level buffer is a ProDataSet object buffer that is not a child in any active data relation. There may be one or more top-level buffers in a ProDataSet object.

Data type: INTEGER

Access: Readable

Applies to: ProDataSet object handle

NUM-VISIBLE-COLUMNS attribute

Returns the number of visible columns in a browse.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget

NUMERIC-DECIMAL-POINT attribute

The character that represents, in formatted text, a number's decimal point.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

NUMERIC-FORMAT attribute

How to interpret commas and periods within numeric values.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

The possible values are "`American`", "`European`" or a character string consisting of the thousands separator followed by the decimal point. This attribute provides the same functionality as the European Numeric Format (`-E`) parameter.

Note: Although NUMERIC-FORMAT remains writable, it accepts only the values "`European`" and "`American`." To change the thousands separator or the decimal point in formatted text, use the new SET-NUMERIC-FORMAT( ) method of the SESSION system handle.

NUMERIC-SEPARATOR attribute

The character that represents, in formatted text, a number's thousands separator.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

ON-FRAME-BORDER attribute

Indicates whether the last event was a mouse event that occurred on a frame border.

Data type: LOGICAL

Access: Readable

Applies to: LAST-EVENT system handle

ORIGIN-HANDLE attribute

Returns the handle of the temp-table in the original source ProDataSet object that corresponds to the temp-table currently associated with this temp-table handle.

Data type: HANDLE

Access: Readable

Applies to: Temp-table object handle

Progress uses this value to match up temp-tables in a MERGE-CHANGES operation.

ORIGIN-ROWID attribute

Returns the ROWID of the row in the original before-image table that corresponds to the row in the change table currently associated with this buffer handle.

Data type: ROWID

Access: Readable

Applies to: Buffer object handle

Progress uses this value to match up temp-table rows in a MERGE-CHANGES operation.

OVERLAY attribute

Indicates whether the frame can overlay other frames on the display.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

If the OVERLAY attribute is TRUE, the frame can overlay any other frame that does not have its TOP-ONLY attribute set to TRUE.

OWNER attribute

The handle of the widget that owns a menu widget.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: MENU widget

For a menu bar, the OWNER attribute returns the window with which the menu bar is associated. For a pop-up menu, the OWNER attribute returns the widget with which the menu is associated.

OWNER-DOCUMENT attribute

Returns the handle of the owning document of a node.

Data type: HANDLE

Access: Readable

Applies to: X-noderef object handle

The following example demonstrates the use of the OWNER-DOCUMENT attribute:

```
DEFINE VARIABLE hDoc as HANDLE.
DEFINE VARIABLE hDoc2 as HANDLE.
hDoc:LOAD("file","my.xml",true).
hDoc:GET-DOCUMENT-ELEMENT(hNoderef).
hDoc2 = hNoderef:OWNER-DOCUMENT.
/* At this point, hDoc2 and hDoc should be the same. */
```

PAGE-BOTTOM attribute

Indicates whether a frame is a footer frame in paged output.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

If PAGE-BOTTOM is TRUE, the frame appears at the end of each page of output.

PAGE-TOP attribute

Indicates whether a frame is a header frame in paged output.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

If PAGE-TOP is TRUE, the frame appears at the beginning of each page of output.

PARAMETER attribute

The value of the Parameter (`-param`) startup parameter specified for the current session.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

Use the Parameter (`-param`) parameter to specify a character string that can be accessed from 4GL procedures. Progress does not check the value of the PARAMETER attribute and you can use the parameter and attribute to store any arbitrary string value.

PARENT attribute

The handle of the parent of a widget.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is read-only for field groups.

For field-level widgets, the parent widget is the field group that contains the widget. For field groups, the parent widget is the frame that contains the field group. For frames, the parent widget is the window or field group that contains the frame. In windows, the parent widget is the window that parents the window. For a submenu or menu item, the parent widget is the menu or submenu that contains the submenu or menu item.

PARENT-BUFFER attribute

Returns the buffer handle of the parent member of the data-relation object.

Data type: HANDLE

Access: Readable

Applies to: Data-relation object handle

PARENT-RELATION attribute

Returns the handle to the SELECTION data-relation object for the parent of this buffer. Since a buffer may be a child in only one relation, there can be only one parent for any buffer.

Data type: HANDLE

Access: Readable

Applies to: Buffer object handle

PARSE-STATUS attribute

The current status of a SAX parse.

Data type: INTEGER

Access: Readable

Applies to: SAX-reader object handle

The default value is SAX-UNINITIALIZED.

The values that PARSE-STATUS can assume are described in Table 75.

Table 75: PARSE-STATUS attribute values

This value...	Indicates...
SAX-UNINITIALIZED	No parsing has occurred.
SAX-RUNNING	Parsing has begun.
SAX-COMPLETE	Parsing has begun and one of the following has occurred: • The parser has determined that there are no more tokens in the XML source. • The application has stopped the parser by calling the STOP-PARSING( ) method.
SAX-PARSER-ERROR	One of the following has occurred: • The parser could not start or could not continue. Perhaps the parser could not be loaded, the XML source could not be found, the XML source was invalid, etc. • The parser started, but a callback executed a RETURN ERROR statement.

PASSWORD-FIELD attribute

Displays password data in a field as a series of fill characters.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FILL-IN widget

If TRUE, the current value of a fill-in field or any character value typed into the fill-in field is displayed as a series of fill characters. The default value is FALSE.

In Windows GUI platforms, the default fill character is the asterisk (*). On non-Windows GUI or character platforms, the default fill character is a blank.

PATHNAME attribute

The absolute or relative pathname of the file specified by the FILE-NAME attribute of the FILE-INFO Handle.

Data type: CHARACTER

Access: Readable

Applies to: FILE-INFO system handle

If the FILE-NAME attribute specifies a simple filename or relative pathname, this attribute returns a relative pathname based on the `PROPATH`. Otherwise, it returns the absolute pathname specified in FILE-NAME.

PBE-HASH-ALGORITHM attribute

A text string containing the name of the hash algorithm to use with the GENERATE-PBE-KEY function to generate a password-based encryption key. The default value is “SHA-1”.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

Progress supports the following hash algorithms:

- United States Government Secure Hash Algorithm (SHA-1).
- RSA Message Digest Hash Algorithm (MD5).

You are responsible for generating, storing, and transporting this value.

See also MD5-DIGEST function, SHA1-DIGEST function

PBE-KEY-ROUNDS attribute

The number of hash algorithm iterations to perform in the GENERATE-PBE-KEY function to generate a password-based encryption key. The value must be a positive integer. The default value is 1000.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

You are responsible for generating, storing, and transporting this value.

Note: Setting the PBE-KEY-ROUNDS attribute to an extremely large number can significantly slow the performance of the GENERATE-PBE-KEY function.

PERSISTENT attribute

Indicates whether the procedure is persistent.

Data type: LOGICAL

Access: Readable/Writable

Applies to: CALL object handle, THIS-PROCEDURE system handle (and all procedure handles)

The PERSISTENT attribute is TRUE when the RUN statement that executes a procedure is invoked with the PERSISTENT option. Otherwise, it is FALSE.

If PERSISTENT is TRUE, when a dynamic invoke returns, the IN-HANDLE attribute contains a handle to the running persistent procedure.

Returns TRUE for a Web service procedure.

PERSISTENT-CACHE-DISABLED attribute

Indicates whether WebClient disables the saving of security cache attribute values between sessions (TRUE) or not (FALSE).

Data type: LOGICAL

Access: Readable

Applies to: CODEBASE-LOCATOR system handle

When TRUE, KEEP-SECURITY-CACHE will be FALSE.

PERSISTENT-PROCEDURE attribute

For the AppServer, this attribute returns the proxy remote persistent procedure handle of the remote procedure that contains the internal procedure executed for the specified asynchronous request. For Web services, this attribute returns the Web service procedure object handle.

Data type: HANDLE

Access: Readable

Applies to: Asynchronous request object handle

This handle is the same as the handle specified by the IN *proc-handle* option of the RUN statement that executes this request. If the request is running a remote external (not internal) procedure, this attribute contains an invalid handle.

PFCOLOR attribute (Character interfaces only)

The color number of the color of a widget that has input focus. The edge color of a rectangle widget.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (cell), BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TOGGLE-BOX widget, WINDOW widget

The color number represents an entry in the color table maintained by the COLOR-TABLE handle.

For field-level widgets that receive focus, the PFCOLOR attribute specifies the input color for the widget. In windows, the PFCOLOR attribute specifies the color inherited by menu items in the menu bar when they are chosen, if the menu items don't already have the PFCOLOR specified.

For browse widgets, this color represents the input color for the focused cell.

For more information on widget color, see the DCOLOR attribute.

PIXELS-PER-COLUMN attribute

The number of pixels in each column of the display.

Data type: INTEGER

Access: Readable

Applies to: SESSION system handle

This value is also the pixel size of a horizontal character unit, and depends on the resolution of the display and the size of the default system font.

PIXELS-PER-ROW attribute

The number of pixels in each row of the display.

Data type: INTEGER

Access: Readable

Applies to: SESSION system handle

This value is also the pixel size of a vertical character unit, and depends on the resolution of the display and the size of the default system font.

POPUP-MENU attribute

The pop-up menu associated with a widget.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget, WINDOW widget

The value you assign to POPUP-MENU must be the handle of a previously defined menu whose POPUP-ONLY attribute is TRUE.

POPUP-ONLY attribute

Indicates whether a menu is pop-up or a menu bar.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: MENU widget

Set the POPUP-ONLY attribute to TRUE to use the menu as a pop-up menu. Otherwise, the menu is a menu bar that you can associate with a window. FALSE is the default value. You can set this attribute only before the menu is realized.

POSITION attribute

The position of a buffer-field within the database record.

Data type: INTEGER

Access: Readable

Applies to: Buffer-field object handle

Note: The POSITION attribute applies to OpenEdge databases only.

PREFER-DATASET attribute

Specifies whether Progress ignores modifications to the data currently in the data source when saving changes from a ProDataSet temp-table buffer to the associated data source using the SAVE-ROW-CHANGES( ) method. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-source object handle

If TRUE, Progress ignores the data currently in the data source and automatically accepts the data from the ProDataSet temp-table buffer.

If FALSE, Progress compares the before-image of the data in the ProDataSet temp-table buffer, saved while tracking changes for the buffer, to the corresponding data source buffer to determine whether the data in the data source has changed since being read. Progress evaluates any data source changes to determine whether or not a data conflict exists based on the MERGE-BY-FIELD attribute setting (that is, on either a field-to-field or buffer-to-buffer basis).

PREPARED attribute

This attribute returns TRUE if the TEMP-TABLE-PREPARE( ) method has been called with no subsequent CLEAR( ) method. That is, it is true when the temp-table is in the PREPARED state.

Data type: LOGICAL

Access: Readable

Applies to: Temp-table object handle

PREPARE-STRING attribute

The character string passed to the most recent QUERY-PREPARE. If QUERY-PREPARE was not called, or the query was just opened with the OPEN QUERY statement, PREPARE-STRING has the Unknown value (?).

Data type: CHARACTER

Access: Readable

Applies to: Query object handle

For an example, see the reference entry for the INDEX-INFORMATION attribute in this book.

PREV-COLUMN attribute

The widget handle of the previous sibling, in physical order, of the current browse column whether or not the column is visible. The browse MOVE-COLUMN method changes the physical order of columns and updates this attribute accordingly.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: BROWSE widget (column)

PREV-SIBLING attribute

The previous entry in the list of handles, relative to a given handle.

Note: Returns the Unknown value (?) for a Web service procedure.

Data type: HANDLE

Access: Readable

Applies to: Asynchronous request object handle, BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, Server object handle, SLIDER widget, Socket object handle, Server socket object handle, SUB-MENU widget, TEXT widget, THIS-PROCEDURE system handle (and all procedure handles), TOGGLE-BOX widget, WINDOW widget

Table 76 summarizes the value of PREV-SIBLING for each relevant handle type.

Table 76: PREV-SIBLING attribute values by handle type *(1 of 2)*

Handle type	Value of NEXT-SIBLING
Asynchronous Request	The handle of the previous asynchronous request submitted for execution on the AppServer or Web Server that is running the specified request.
Procedure	The handle of the previous persistent procedure in the current OpenEdge session. If the current procedure is a proxy for a persistent procedure running on an AppServer or for a Web service, specifies the previous procedure object bound to the same server handle.
Server	The previous server handle created in the current OpenEdge session (independent of subtype).

Table 76: PREV-SIBLING attribute values by handle type *(2 of 2)*

Handle type	Value of NEXT-SIBLING
Socket and Server-socket	The previous socket handle in the chain of socket handles for the current OpenEdge session. Returns the Unknown value (?) for the first handle in the chain.
Widget	The handle of the previous widget in the widget list. **Note:** A widget must first be realized before it can become part of the list. A hidden widget cannot become part of the list since it is not realized. A widget that is already part of the list can be hidden and it remains part of the list.

If the given handle is the first handle in the list, PREV-SIBLING assumes the value of an invalid handle. To check the validity of a handle, use the VALID-HANDLE function.

See also NEXT-SIBLING attribute

PREV-TAB-ITEM attribute

The widget handle of the previous widget in the tab order of a field group relative to the specified widget.

Data type: HANDLE

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

The PREV-TAB-ITEM attribute returns the Unknown value (?) for a widget that is at the beginning of the tab order in a field group.

PRIMARY attribute

This attribute sets or returns the name of the temp-table's primary index. PRIMARY can only be updated before the TEMP-TABLE-PREPARE( ) method has been called. It returns the Unknown value (?) if the temp-table is not in a PREPARED state.

Data type: CHARACTER

Access: Readable/Writable

Applies to: Temp-table object handle

PRINTER-CONTROL-HANDLE attribute (Windows only)

The default context for print jobs.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

The print context is an integer identifier for a set of values that define a printer and setup for that printer in Windows. You can establish a print context using the Print dialog box. The SYSTEM-DIALOG PRINTER-SETUP statement allows you to display the Print dialog box and set a print context. This print context is used by the OUTPUT TO PRINTER statement to direct output to a printer.

If the PRINTER-CONTROL-HANDLE attribute contains zero (0) or the Unknown value (?), the print context is the default print context in Windows as set in the Windows Control Panel. You can assign any integer value to this attribute, but the result of the assignment will be to set the attribute value to 0 and to release any Windows resources related to the previous print context.

PRINTER-HDC attribute

A handle to the current Windows device context for a print job.

Data type: INTEGER

Access: Readable

Applies to: SESSION system handle

The printer device context handle is the Windows Handle to a Device Context (HDC). The value of the PRINTER-HDC attribute is meaningless when the SESSION:PRINTER-CONTROL-HANDLE has a value of zero (0) or the Unknown value (?). For more information, see the reference entry for the PRINTER-CONTROL-HANDLE attribute in this book.

PRINTER-NAME attribute (Windows only)

The name of the currently selected printer in Windows platforms, and the Unknown value (?) on other platforms.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

Use this attribute to set the printer name in the default print context. The value of PRINTER-NAME is the name Windows uses to identify a printer. The specified printer must be defined in the Windows Registry. If the specified printer is not defined in the Windows Registry, the value of PRINTER-NAME is not modified. You must specify network printers in Universal Naming Convention format.

Use the GET-PRINTERS( ) method to get the list of printers currently defined in the Windows Registry.

If you use the SYSTEM-DIALOG PRINTER-SETUP statement to set the printer name, this attribute assumes the modified value.

PRINTER-PORT attribute (Windows only)

The currently selected printer port in Windows platforms, and the Unknown value (?) on other platforms.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

PRINTER-PORT assumes the value for printer port that Windows defines. If someone modifies the value using the SYSTEM-DIALOG PRINTER-SETUP command, PRINTER-PORT assumes the modified value.

PRIVATE-DATA attribute

An arbitrary string associated with the handle of an object or widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Asynchronous request object handle, BROWSE widget (browse and column), Buffer-field object handle, Buffer object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, ProDataSet object handle, Data-relation object handle, Data-source object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, Query object handle, RADIO-SET widget, RECTANGLE widget, SAX-attributes object handle, SAX-reader object handle, SELECTION-LIST widget, Server object handle, Server socket object handle, SLIDER widget, SOAP-header object handle, SOAP-header-entryref object handle, Socket object handle, SUB-MENU widget, TEXT widget, THIS-PROCEDURE system handle (and all procedure handles), TOGGLE-BOX widget, WINDOW widget

Use this attribute any way you want. Progress does not check the value of this attribute.

PROCEDURE-NAME attribute

A string specifying the name of the remote procedure executed to instantiate the specified asynchronous request handle.

Data type: CHARACTER

Access: Readable

Applies to: Asynchronous request object handle

This name is the same as the *extern-proc-name*, *intern-proc-name*, or VALUE option used to specify the remote procedure executed in the asynchronous RUN statement.

PROGRESS-SOURCE attribute (Character interfaces only)

How an editor widget wraps lines of 4GL source code that are longer than the widget's display width.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

Set this attribute to TRUE when reading Progress 4GL source code into the widget. This preserves the 4GL syntax for the Progress compiler.

When set to TRUE, the editor widget splits long lines by putting a tilde (~) and hard return at the end of the display line and continuing the text with column 1 of the next display line. The line wrapping occurs only when a READ-FILE( ) or INSERT-FILE( ) method is used to bring text into the widget. When set to FALSE, the editor widget splits long lines by inserting a HARD-RETURN before the last word and moving the last word onto the next display line. FALSE is the default setting.

PROXY attribute

Indicates whether a procedure handle is a proxy persistent procedure handle.

Data type: LOGICAL

Access: Readable

Applies to: THIS-PROCEDURE system handle (and all procedure handles)

If PROXY is TRUE, the procedure handle is a proxy handle for a persistent procedure running remotely in the context of an AppServer.

PROXY is always FALSE on the THIS-PROCEDURE handle by definition.

Returns TRUE for a Web service procedure.

For more information on the AppServer, see *OpenEdge Application Server: Developing AppServer Applications*.

PROXY-PASSWORD attribute

Authenticates an AppServer or Web service client to the HTTP-based proxy server.

Data type: Character

Access: Readable/Writeable

Applies to: SESSION system handle

This attribute corresponds to the `-proxyPassword` startup parameter. If `-proxyPassword` is not specified on the command line, this attribute has the Unknown value (?) until it is set.

This attribute is validated during the CONNECT( ) method (for an AppServer and a Web service). If PROXY-PASSWORD is invalid, the CONNECT( ) method fails and Progress issues an error message. If PROXY-USERID **is not** unknown and PROXY-PASSWORD **is** unknown, the AppServer CONNECT( ) method uses a blank proxy password. If SESSION:PROXY-USERID is unknown, the CONNECT( ) method ignores any value of PROXY-PASSWORD. PROXY-PASSWORD must be a string of up to 512 printable ASCII characters.

See also PROXY-USERID attribute, *OpenEdge Deployment: Startup Command and Parameter Reference*, *OpenEdge Deployment: WebClient Applications*

PROXY-USERID attribute

Authenticates an AppServer or Web service client to the HTTP-based proxy server.

Data type: Character

Access: Readable/Writeable

Applies to: SESSION system handle

This attribute corresponds to the `-proxyUserid` startup parameter. If `-proxyUserid` is not specified on the command line, this attribute has the Unknown value (`?`) until it is set.

This attribute is validated during the CONNECT( ) method (for an AppServer and a Web service). If PROXY-USERID is invalid, the CONNECT( ) method fails and Progress issues an error message.

PROXY-USERID must be a string of up to 512 printable ASCII characters, including the space character.

See also PROXY-USERID attribute, *OpenEdge Deployment: Startup Command and Parameter Reference*, *OpenEdge Deployment: WebClient Applications*

PUBLIC-ID attribute

This attribute returns the public ID of the external DTD from which an XML document was generated.

Data type: CHARACTER

Access: Readable

Applies to: X-document object handle

PUBLISHED-EVENTS attribute

A comma-separated list of Progress named events published by a particular procedure. Returns the empty string for a Web service procedure.

Note: Progress named events are completely different from the key function, mouse, widget, and direct manipulation events described in the "Events Reference" section on page 2171. For more information on Progress named events, see *OpenEdge Development: Progress 4GL Handbook*.

Data type: CHARACTER

Access: Readable

Applies to: THIS-PROCEDURE system handle (and all procedure handles)

Progress builds PUBLISHED-EVENT lists as the compiler encounters PUBLISH statements-specifically, PUBLISH statements that specify the event name as a quoted string and not as a CHARACTER variable expression, and that do not use the FROM option.

Note: PUBLISHED-EVENTS lists do not contain signatures of their named events. Progress assumes that the subscriber to a named event knows its signature.

QUERY attribute

The handle of the query connected to a browse widget, a buffer object, a data-relation object, or a data-source object.

Data type: WIDGET-HANDLE

Access: Readable/Writable

Applies to: BROWSE widget, Buffer object handle, Data-relation object handle, Data-source object handle

If you change the value of a browse's QUERY attribute, you connect the browse to a different query, which contains a different set of records.

For a browse query in Windows platforms:

- The original query and the new query do not need to have the same underlying database fields.
- If the query is changed for a dynamic browse, the browse columns are removed. You should add new columns with the ADD-CALC-COLUMN, ADD-COLUMNS-FROM, and ADD-LIKE-COLUMN methods.
- If the query is changed for a static browse and the underlying fields are the same, the columns are not removed. However, if the underlying fields are not the same, the columns are removed. The columns are also removed if the QUERY attribute is set to the Unknown value (?). You should add new columns with the ADD-CALC-COLUMN, ADD-COLUMNS-FROM, and ADD-LIKE-COLUMN methods.
- Also, a query can now be attached to a static browse that was defined without the optional DISPLAY phrase.

For a browse query on Character Mode platforms:

- If the original query has database tables, the new query must have database tables. The new query can have different buffers as long as they correspond to the same database tables.
- If the original query has temp-tables, the new query must have temp-tables, not work tables.
- The original query and the new query must have the same number of tables.

For a buffer object, this attribute returns the handle to the query currently associated with the buffer (if any). If the buffer does not have an associated query, this attribute returns the Unknown value (?). This attribute is also read-only for a buffer object.

For a data-relation object, this attribute returns the handle to the default dynamic query for a child buffer in the relation. This automatically generated query expresses the relation between parent and child temp-tables, and lets you navigate the child records. This handle cannot be set, and the query cannot be modified.

For a data-source object, this attribute associates a query with a dynamic Data-source object. To disassociate the query and Data-source object, set this attribute to the Unknown value (?). You can also use the FILL-WHERE-STRING attribute to override the WHERE clause in the query.

See also FILL-WHERE-STRING attribute

QUERY-CLOSE( ) method

Closes a query object.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
QUERY-CLOSE ( )
```

Note: A QUERY-CLOSE does not invalidate a previous QUERY-PREPARE.

QUERY-OFF-END attribute

Indicates whether a query is positioned off either end of its result list (that is, either before the first record or after the last record).

Data type: LOGICAL

Access: Readable

Applies to: Query object handle

The QUERY-OFF-END attribute corresponds to the QUERY-OFF-END function.

Note: Progress also provides an OFF-END event for when a query on a ProDataSet temp-table buffer is positioned past the last row. You can use this event to retrieve additional data source rows to add at the bottom of a ProDataSet temp-table (for example, in batches when there are too many data source rows to retrieve at one time). The OFF-END event is similar to the QUERY-OFF-END attribute, which is set to TRUE whenever the associated query object is positioned past the last row. The difference is that you must test the QUERY-OFF-END attribute for this condition at a specific place in your application code, whereas the OFF-END event procedure executes like a trigger whenever the event occurs. For more information about the OFF-END event on a ProDataSet temp-table query, see the "ProDataSet events" section on page 2195.

See also QUERY-OFF-END function

QUERY-OPEN( ) method

Opens a query object.

Note: You must perform QUERY-PREPARE on a query object before you perform QUERY-OPEN on it.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
QUERY-OPEN ( )
```

Once you perform QUERY-PREPARE on a query object, you can perform QUERY-OPEN on it multiple times as long as you do not reperform QUERY-PREPARE. Once you reperform QUERY-PREPARE, you must reperform QUERY-OPEN.

Note: The QUERY-PREPARE and QUERY OPEN methods correspond to the OPEN QUERY statement.

QUERY-PREPARE( ) method

Compiles a predicate (query condition).

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
QUERY-PREPARE ( predicate-expression )
```

predicate-expression

A CHARACTER expression that evaluates to an OPEN QUERY **. . .** FOR EACH statement without the OPEN QUERY **. . .**. You can also use a field phrase.

The QUERY-PREPARE method corresponds to the OPEN QUERY statement's compilation phase. To open the query object, use the QUERY-OPEN method.

If the QUERY-PREPARE method encounters an error, it returns FALSE and does not raise an error.

If you use the QUERY-PREPARE method in a statement that uses the NO-ERROR option and an error occurs, the QUERY-PREPARE method still returns FALSE and does not raise an error. You can get information on the error through the GET-MESSAGE method of the ERROR-STATUS system handle, as usual.

Note: The QUERY-PREPARE method is compatible with indexed reposition of queries with joins. In *predicate-expression*, just include the INDEXED-REPOSITION option. For more information on the INDEXED-REPOSITION option, see the reference entry for the OPEN QUERY statement.

The following are examples:

```
my-query-handle:QUERY-PREPARE("for each customer where cust-num < 9)".
```

```
my-query-handle:QUERY-PREPARE(my-predicate).
```

```
my-query-handle:QUERY-PREPARE("for each customer where cust-num > "
  + string(my-integer)).
```

QUIT attribute

Indicates that a QUIT condition was returned from the AppServer as a result of processing the specified asynchronous request. Returns FALSE for an asynchronous request made on a Web service.

Data type: LOGICAL

Access: Readable

Applies to: Asynchronous request object handle

If the COMPLETE attribute is FALSE, the value of this attribute is the Unknown value (?). When the PROCEDURE-COMPLETE event is processed, this attribute is set to TRUE before the event procedure is executed if the remote request returned with an unhandled QUIT condition; otherwise, it is set to FALSE.

RADIO-BUTTONS attribute

The label and value associated with each radio button in a radio set.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: RADIO-SET widget

You can set this attribute to a comma-separated list containing the label/value pairs associated with each button. Each label and each value should be followed by a comma, as in "label1,value1,label2,value2,...label*n*,value*n*".

RAW-TRANSFER( ) method

Copies data to or from a buffer object with no interpretation. This method works like the RAW-TRANSFER statement.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
bh:RAW-TRANSFER ( to-mode , handle-expression )
```

to-mode

A logical specifying the direction of the data transfer. When *to-mode* is TRUE, data is transferred from *bh* to *handle-expression*. When *to-mode* is FALSE, data is transferred from *handle-expression* to *bh*.

handle-expression

An expression that evaluates to the handle of either a buffer or a buffer field.

When using the RAW-TRANSFER statement to copy from a buffer object that contains a BLOB or CLOB field, Progress skips the BLOB or CLOB field and stores the Unknown value (?) in the BLOB or CLOB field of the target buffer object.

See also RAW-TRANSFER statement

READ( ) method

Reads data from the socket.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
READ( buffer , position , bytes-to-read , [ mode ] )
```

buffer

A MEMPTR expression that identifies where the data which is read from the socket should be stored.

position

An INTEGER expression greater than 0 that indicates the starting byte position within *buffer* into which information should be written.

bytes-to-read

An INTEGER expression that specifies the number of bytes to be read from the socket.

mode

An optional INTEGER expression that specifies how *bytes-to-read* should be interpreted. Table 77 shows the valid values for this parameter. The default value is READ-EXACT-NUM (2).

Table 77: Valid read modes for the READ( ) method

Compiler constant	Value	Description
READ-AVAILABLE	1	The READ( ) method will block until at least one byte has been read on the socket. It will read up to *bytes-to-read* bytes.
READ-EXACT-NUM	2	The READ( ) method will block until *bytes-to-read* bytes have been read from the socket.

READ( ) returns TRUE if the read operation succeeded normally and returns FALSE otherwise. An error can occur if:

- The position parameter is not greater than 0.
- Amount of information requested to read exceeds the size of buffer.
- Reading from the socket fails.

This read statement is a blocking read. If *mode* is READ-EXACT-NUM, this method returns when it has either read the requested number of bytes from the socket or an error occurs. If *mode* is READ-AVAILABLE, this method returns when it has read as many bytes as are currently available on the socket, up to the requested number of bytes, or an error occurs.

If the READ( ) method succeeds, the variable buffer contains the data which is read from the socket. It is possible that the socket will not contain the specified number of bytes of data which were requested. The BYTES-READ attribute can be used to determine the number of bytes read from the socket.

This method expects *buffer* to identify a MEMPTR variable which already has a region of memory associated with it. The developer must call the SET-SIZE statement to allocate memory and associate it with a MEMPTR variable. It is the responsibility of the developer to free this memory, also via the SET-SIZE statement. The READ method will fail if the size of *buffer* is less than *bytes-to-read*.

READ-FILE( ) method

Clears an editor widget, reads the contents of a specified text file into the widget, and sets the widget's MODIFIED attribute to FALSE.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
READ-FILE ( filename )
```

filename

A character-string expression of a full or relative pathname of a file.

The READ-FILE( ) method searches the PROPATH to find the file. If the operation is successful, the method returns TRUE.

In Windows, this method interprets a carriage return character followed by a line feed character as a text line terminator. In all other interfaces, this method interprets a carriage return character as a text line terminator.

READ-ONLY attribute

Indicates whether an object is write-protected.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), Buffer-field object handle, EDITOR widget, FILL-IN widget, MENU-ITEM widget

If the READ-ONLY attribute of an editor or fill-in widget is TRUE, the widget cannot be enabled for input and its screen value cannot be changed from the user interface.

If the READ-ONLY attribute for a browse widget is TRUE, you cannot update editable cells. If it is set to false, then the ability to edit cells is restored. This functionality only applies to columns that have been enabled in the DEFINE BROWSE statement.

If the READ-ONLY attribute of a menu item is TRUE, the menu item cannot be chosen. You can set this attribute for a menu item only before the widget is realized.

The READ-ONLY attribute has no effect on the appearance of a widget. If an editor or menu item is insensitive, it is grayed out in some environments. Use the READ-ONLY attribute instead to make the widget insensitive without being grayed out.

READ-XML( ) method

Reads an XML document into a ProDataSet, temp-table, or temp-table buffer object. You can read data, schema, or both.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
READ-XML ( source-type, { file | memptr | handle | longchar }, read-mode,
schema-location, override-default-mapping [, field-type-mapping [,
verify-schema-mode ] ] )
```

source-type

A CHARACTER expression that specifies the source XML document type. Valid values are: “FILE”, “MEMPTR”, “HANDLE”, and “LONGCHAR”.

file

A CHARACTER expression that specifies the name of a file. You can specify an absolute pathname, a relative pathname (based on the current working directory), or a URL pathname. Valid URL protocols include FILE and HTTP (the HTTPS protocol is not supported). Progress verifies that the file exists and is accessible.

memptr

A MEMPTR variable that contains the XML document text in memory. The size of the MEMPTR variable must match the size of the XML document text.

handle

A WEB-CONTEXT system handle, X-document object handle, or X-noderef object handle.

For a WEB-CONTEXT system handle, the READ-XML( ) method reads an XML document from the WebSpeed transaction server. The method verifies that the XML document was posted to the WebSpeed transaction server (that is, the value of the IS-XML attribute for the handle is YES), and that Progress is running in a WebSpeed environment.

longchar

A LONGCHAR variable that contains the XML document text in memory.

read-mode

A CHARACTER expression that specifies the mode in which the READ-XML( ) method reads data from the XML document into a temp-table or ProDataSet member buffer. The expression must evaluate to “APPEND”, “EMPTY”, “MERGE”, or “REPLACE”. The default value is "MERGE".

Table 78 lists the READ-XML( ) method modes for reading data.

Table 78: READ-XML( ) method read modes

When the mode is ...	The READ-XML( ) method ...
APPEND	Reads data from the XML document into the ProDataSet or temp-table object by adding new records to the existing records, without performing any record comparisons. If a record from the XML document exists in the object (that is, it results in a duplicate unique key conflict), the method generates an error message and returns FALSE.
EMPTY	Empties the contents of the ProDataSet or temp-table object before reading in data from the XML document.
MERGE	Reads data from the XML document into the ProDataSet or temp-table object by merging new records with existing records in the table. If a record from the XML document exists in the object (that is, it results in a duplicate unique key conflict), the method does not replace the existing record. If the record from the XML document does not exist in the object, the method creates a new record.
REPLACE	Reads data from the XML document into the ProDataSet or temp-table object by merging new records with existing records in the table. If the record from the XML document exists in the object (that is, it results in a duplicate unique key conflict), the method replaces the existing record with the new record. If the record from the XML document does not exist in the object, the method creates a new record.

schema-location

A CHARACTER expression that specifies the name of an external XML Schema file to use in creating or verifying the object's schema when reading in the XML document. You can specify an absolute pathname, a relative pathname (based on the current working directory), or a URL pathname. Valid URL protocols include FILE and HTTP (the HTTPS protocol is not supported). Progress verifies that the file exists and is accessible. When specified, Progress ignores any schema defined or referenced in the source XML Document.

If you specify the empty string ("") or the Unknown value (?), Progress creates or verifies the object's schema using any XML Schema defined or referenced in the XML document.

override-default-mapping

A LOGICAL expression where TRUE directs Progress to override the default mapping between XML Schema string and binary data types and Progress data types when creating a Progress temp-table schema from an XML Schema. The default value is FALSE.

The XML Schema string data type maps to the Progress CHARACTER data type by default, and the XML Schema base64Binary and hexBinary data types map to the Progress RAW data type by default. If you specify TRUE, the READ-XML( ) method creates a temp-table schema with CLOB and BLOB fields instead of CHARACTER and RAW fields.

If you specify the Unknown value (?), the method uses the default value of FALSE.

field-type-mapping

An optional CHARACTER expression that evaluates to a comma-separated list of field name, data type pairs using the following syntax:

```
field-name-1, data-type-1 [, field-name-n, data-type-n ] ...)
```

This option allows you to specify the Progress data type for a specific field from the XML Schema.

field-name

A CHARACTER expression that evaluates to the name of the specified field. For a ProDataSet object, you must qualify the field name with the buffer name from the XML Schema. That is, *buffer-name.field-name*.

data-type

A CHARACTER expression that evaluates to the data type of the specified field. The data type must be a valid Progress data type, and it must be compatible with the XML Schema type based on the Progress XML data type mapping rules. For example, any XML Schema type can be mapped to a Progress CHAR or CLOB, but an XML Schema dateTime can be mapped only to a Progress DATE, DATETIME or DATETIME-TZ.

If you specify the Unknown value (?), the method uses the default data type mapping. For more information about the Progress XML data type mapping rules, see *OpenEdge Development: Programming Interfaces*.

verify-schema-mode

An optional CHARACTER expression that specifies the mode in which the READ-XML( ) method verifies any XML Schema against existing Progress schema. The expression must evaluate to “IGNORE”, “LOOSE”, or “STRICT”. The default value is “LOOSE”.

Note: For a dynamic temp-table or ProDataSet member buffer that does not have Progress schema (that is, the object is in the CLEAR state), this option is ignored.

Table 79 lists the READ-XML( ) method schema verification modes.

Table 79: READ-XML( ) method schema verification modes *(1 of 2)*

When the mode is ...	The READ-XML( ) method ...
IGNORE	Ignores any XML Schema specified in *schema-location*, or defined in the XML Document.
LOOSE	For temp-table objects: Matches temp-table columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. The XML Schema may be a subset or superset of the temp-table schema. Any columns that are in the XML Schema but not in the temp-table are ignored. Any columns that are in the temp-table, but not in the XML Schema, are ignored. For ProDataSet objects: Matches temp-tables and columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. Data relationships are matched by parent buffer and child buffer names. For every data relationship in the XML Schema that matches a data-relation in the ProDataSet, the field mapping between the parent and child buffers must match. The XML Schema may be a subset or superset of the ProDataSet schema. Any temp-tables, columns, or data-relations that are in the ProDataSet, but not in the XML Schema, are ignored. For a dynamic ProDataSet object, the method adds temp-tables and data-relations to the object when the temp-tables and data-relations are defined in the XML Schema, but are not members of the ProDataSet. Fields are not added to existing temp-tables. For a static ProDataSet object, any temp-tables or data-relations that are in the XML Schema, but not in the ProDataSet, are ignored.

Table 79: READ-XML( ) method schema verification modes *(2 of 2)*

When the mode is ...	The READ-XML( ) method ...
STRICT	For temp-table objects: Matches temp-table columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. The XML Schema must define a field for every temp-table field, and there can be no extra fields in the XML Schema that are not in the temp-table schema. For ProDataSet objects: Matches temp-tables and columns by name. There must be a temp-table defined in the XML Schema for each table of the ProDataSet. There can be no tables defined in the XML Schema that are not in the ProDataSet schema. There must be field defined in the XML Schema for each field in the temp-table, with the same data type and extent, and there can be no fields defined in the XML Schema that are not in the temp-table schema. There must also be a data relationship defined in the XML Schema for every data-relation in the ProDataSet, and there can be no data relationships defined in the XML Schema that are not defined in the ProDataSet. The field mapping between the parent and child buffers must match.

If you specify the Unknown value (?), the method uses the default value of LOOSE.

If the XML Schema verification fails, the method generates an error message indicating the XML Schema element that caused the failure and returns FALSE.

Examples

The following code example creates a dynamic ProDataSet object from an empty ProDataSet handle, creates the object's schema from the specified XML Schema file, and populates the temp-tables with records from the specified XML document:

```
DEFINE VARIABLE cSourceType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cReadMode AS CHARACTER NO-UNDO.
DEFINE VARIABLE lOverrideDefaultMapping AS LOGICAL NO-UNDO.
DEFINE VARIABLE cFile AS CHARACTER NO-UNDO.
DEFINE VARIABLE cEncoding AS CHARACTER NO-UNDO.
DEFINE VARIABLE cSchemaLocation AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFieldTypeMapping AS CHARACTER NO-UNDO.
DEFINE VARIABLE cVerifySchemaMode AS CHARACTER NO-UNDO.
DEFINE VARIABLE retOK AS LOGICAL NO-UNDO.
DEFINE VARIABLE hDSet AS HANDLE NO-UNDO.

CREATE DATASET hDSet.
ASSIGN
     cSourceType = "file"
     cFile = "dset.xml"
     cReadMode = "empty"
     cSchemaLocation = "cust-ord-inv.xsd"
     lOverrideDefaultMapping = ?
     cFieldTypeMapping = ?
     cVerifySchemaMode = ?.

retOK = hDSet:READ-XML(cSourceType,
                       cFile,
                       cReadMode,
                       cSchemaLocation,
                       lOverrideDefaultMapping,
                       cFieldTypeMapping,
                       cVerifySchemaMode).
```

The following code example creates a dynamic temp-table object from an empty temp-table handle, creates the object's schema from the specified XML Schema file, and populates the temp-table with records from the specified XML document:

```
DEFINE VARIABLE cSourceType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cReadMode AS CHARACTER NO-UNDO.
DEFINE VARIABLE lOverrideDefaultMapping AS LOGICAL NO-UNDO.
DEFINE VARIABLE cFile AS CHARACTER NO-UNDO.
DEFINE VARIABLE cEncoding AS CHARACTER NO-UNDO.
DEFINE VARIABLE cSchemaLocation AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFieldTypeMapping AS CHARACTER NO-UNDO.
DEFINE VARIABLE cVerifySchemaMode AS CHARACTER NO-UNDO.
DEFINE VARIABLE retOK AS LOGICAL NO-UNDO.
DEFINE VARIABLE httCust AS HANDLE NO-UNDO.

CREATE TEMP-TABLE httCust.
ASSIGN
     cSourceType = "file"
     cFile = "ttcust.xml"
     cReadMode = "empty"
     cSchemaLocation = "ttcust.xsd"
     lOverrideDefaultMapping = ?
     cFieldTypeMapping = ?
     cVerifySchemaMode = ?.

retOK = httCust:READ-XML(cSourceType,
                         cFile,
                         cReadMode,
                         cSchemaLocation,
                         lOverrideDefaultMapping,
                         cFieldTypeMapping,
                         cVerifySchemaMode).
```

Notes

- If the ProDataSet or temp-table object does not have a schema (that is, the object is dynamic and in the CLEAR state), Progress creates the schema from either the XML Schema file specified in *schema-location*, or the XML Schema defined or referenced in the XML document. If a dynamic temp-table is not in the PREPARED or CLEAR state, the method generates an error and returns FALSE.

  If the ProDataSet or temp-table object already has a schema (that is, the object is static, or the temp-tables are in the PREPARED state), Progress verifies any XML Schema specified by *schema-location*, or defined or referenced in the XML document, against the object's schema, unless the *verify-schema-mode* is "IGNORE".

If Progress cannot identify any XML Schema for the ProDataSet or temp-table object, (that is, *schema-location* is the empty string ("") or the Unknown value (?) and the XML document does not define or reference a schema, Progress infers the schema from the data in the XML document.

For more information about creating schema from XML Schema, verifying XML Schema, or inferring schema from XML document text, see *OpenEdge Development: Programming Interfaces*.

- The XML document can also contain before-image table data associated with a ProDataSet object. If the XML document data is in the Microsoft DiffGram format, the method reads the before-image data as well. In this case, if the ProDataSet or temp-table object is static and it does not have a before-image table defined, the method generates an error and returns FALSE. If the ProDataSet or temp-table object is dynamic, the method creates the before-image table automatically.

Note: During the read operation, Progress does not respond to ProDataSet events, and it does not track changes to the data in the ProDataSet or temp-table object (that is, it does not update the before-image tables) unless the XML document data is in the Microsoft DiffGram format.

- You cannot read an XML document into a database buffer.

See also IS-XML attribute, READ-XMLSCHEMA( ) method, WEB-CONTEXT system handle, WRITE-XML( ) method, WRITE-XMLSCHEMA( ) method, X-document object handle, X-noderef object handle

READ-XMLSCHEMA( ) method

Reads XML Schema from an XML document and uses that schema to either create a schema for a ProDataSet or temp-table object, or verify existing schema in a ProDataSet, temp-table, or temp-table buffer object. The XML document must be an XML Schema written in the XML Schema Definition (XSD) language in the 2001 XML Schema namespace (http://www.w3.org/2001/XMLSchema).

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
READ-XMLSCHEMA ( source-type, { file | memptr | handle | longchar },
override-default-mapping [, field-type-mapping [, verify-schema-mode ] ] )
```

source-type

A CHARACTER expression that specifies the source XML document type. Valid values are: “FILE”, “MEMPTR”, “HANDLE”, and “LONGCHAR”.

file

A CHARACTER expression that specifies the name of an XML Schema file. You can specify an absolute pathname, a relative pathname (based on the current working directory), or a URL pathname. Valid URL protocols include FILE and HTTP (the HTTPS protocol is not supported). Progress verifies that the file exists and is accessible.

memptr

A MEMPTR variable that contains the XML Schema document text. The size of the MEMPTR variable must match the size of the XML document text.

handle

A WEB-CONTEXT system handle, X-document object handle, or X-noderef object handle.

For a WEB-CONTEXT system handle, the READ-XMLSCHEMA( ) method reads an XML Schema document from the WebSpeed transaction server. The method verifies that the XML document was posted to the WebSpeed transaction server (that is, the value of the IS-XML attribute for the handle is YES), and that Progress is running in a WebSpeed environment.

longchar

A LONGCHAR variable that contains the XML Schema document text in memory.

override-default-mapping

A LOGICAL expression where TRUE directs Progress to override the default mapping between XML Schema string and binary data types and Progress data types when creating a Progress temp-table schema from an XML Schema. The default value is FALSE.

The XML Schema string data type maps to the Progress CHARACTER data type by default, and the XML Schema base64Binary and hexBinary data types map to the Progress RAW data type by default. If you specify TRUE, the READ-XMLSCHEMA( ) method creates a temp-table schema with CLOB and BLOB fields instead of CHARACTER and RAW fields.

If you specify the Unknown value (?), the method uses the default value of FALSE.

field-type-mapping

An optional CHARACTER expression that evaluates to a comma-separated list of field name, data type pairs using the following syntax:

```
field-name-1, data-type-1 [, field-name-n, data-type-n ] ...)
```

This option allows you to specify the Progress data type for a specific field from the XML Schema.

field-name

A CHARACTER expression that evaluates to the name of the specified field. For a ProDataSet object, you must qualify the field name with the buffer name from the XML Schema. That is, *buffer-name.field-name*.

data-type

A CHARACTER expression that evaluates to the data type of the specified field. The data type must be a valid Progress data type, and it must be compatible with the XML Schema type based on the Progress XML data type mapping rules. For example, any XML Schema type can be mapped to a Progress CHAR or CLOB, but an XML Schema dateTime can be mapped only to a Progress DATE, DATETIME or DATETIME-TZ.

If you specify the Unknown value (?), the method uses the default data type. For more information about the Progress XML data type mapping rules, see *OpenEdge Development: Programming Interfaces*.

verify-schema-mode

An optional CHARACTER expression that specifies the mode in which the READ-XMLSCHEMA( ) method verifies any XML Schema against existing Progress schema. The expression must evaluate to “LOOSE” or “STRICT”. The default value is “LOOSE”.

Note: For a dynamic temp-table or ProDataSet member buffer that does not have Progress schema (that is, the object is in the CLEAR state), this option is ignored.

Table 80 lists the READ-XMLSCHEMA( ) method schema verification modes.

Table 80: READ-XMLSCHEMA( ) method verification modes *(1 of 2)*

When the mode is ...	The READ-XMLSCHEMA( ) method ...
LOOSE	For temp-table objects: Matches temp-table columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. The XML Schema may be a subset or superset of the temp-table schema. Any columns that are in the XML Schema but not in the temp-table are ignored. Any columns that are in the temp-table, but not in the XML Schema, are ignored. For ProDataSet objects: Matches temp-tables and columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. Data relationships are matched by parent buffer and child buffer names. For every data relationship in the XML Schema that matches a data-relation in the ProDataSet, the field mapping between the parent and child buffers must match. The XML Schema may be a subset or superset of the ProDataSet schema. Any temp-tables, columns, or data-relations that are in the ProDataSet, but not in the XML Schema, are ignored. For a dynamic ProDataSet object, the method adds temp-tables and data-relations to the object when the temp-tables and data-relations are defined in the XML Schema, but are not members of the ProDataSet. Fields are not added to existing temp-tables. For a static ProDataSet object, any temp-tables or data-relations that are in the XML Schema, but not in the ProDataSet, are ignored.

Table 80: READ-XMLSCHEMA( ) method verification modes *(2 of 2)*

When the mode is ...	The READ-XMLSCHEMA( ) method ...
STRICT	For temp-table objects: Matches temp-table columns by name. The data type and extent of the column in the XML Schema must match those for the matching column in the temp-table. Other field attributes in the XML Schema are ignored. The XML Schema must define a field for every temp-table field, and there can be no extra fields in the XML Schema that are not in the temp-table schema. For ProDataSet objects: Matches temp-tables and columns by name. There must be a temp-table defined in the XML Schema for each table of the ProDataSet. There can be no tables defined in the XML Schema that are not in the ProDataSet schema. There must be field defined in the XML Schema for each field in the temp-table, with the same data type and extent, and there can be no fields defined in the XML Schema that are not in the temp-table schema. There must also be a data relationship defined in the XML Schema for every data-relation in the ProDataSet, and there can be no data relationships defined in the XML Schema that are not defined in the ProDataSet. The field mapping between the parent and child buffers must match.

If you specify the Unknown value (?), the method uses the default value of LOOSE.

If the XML Schema verification fails, the method generates an error message indicating the XML Schema element that caused the failure and returns FALSE.

Examples The following code example verifies the schema in a static ProDataSet object, in STRICT mode, using the schema defined in the specified XML Schema file:

```
DEFINE TEMP-TABLE ttCust LIKE customer.
DEFINE TEMP-TABLE ttOrd  LIKE order.
DEFINE TEMP-TABLE ttInv  LIKE invoice.

DEFINE DATASET DSET FOR ttCust, ttOrd, ttInv
  DATA-RELATION CustOrd FOR ttCust,
    ttOrd RELATION-FIELDS(cust-num,cust-num) NESTED
  DATA-RELATION OrdInv FOR ttOrd,
    ttInv RELATION-FIELDS(order-num,order-num) NESTED.

DEFINE VARIABLE retOK AS LOGICAL NO-UNDO.
DEFINE VARIABLE cSourceType AS CHARACTER  NO-UNDO.
DEFINE VARIABLE cFile AS CHARACTER  NO-UNDO.
DEFINE VARIABLE lOverrideDefaultMapping AS LOGICAL NO-UNDO.
DEFINE VARIABLE cFieldTypeMapping AS CHARACTER NO-UNDO.
DEFINE VARIABLE cVerifySchemaMode AS CHARACTER NO-UNDO.

ASSIGN
    cSourceType = "file"
    cFile = "cust-ord-inv.xsd"
    lOverrideDefaultMapping = NO
    cFieldTypeMapping = ?
    cVerifySchemaMode = "strict".

retOK = DATASET DSET:READ-XMLSCHEMA (cSourceType,
                                     cFile,
                                     lOverrideDefaultMapping,
                                     cFieldTypeMapping,
                                     cVerifySchemaMode).
```

The following code example creates a dynamic temp-table object, creates the object's schema from the specified XML Schema file, and overrides the default data type mapping of one field:

```
DEFINE VARIABLE hTable AS HANDLE NO-UNDO.

CREATE TEMP-TABLE hTable.

DEFINE VARIABLE retOK AS LOGICAL NO-UNDO.
DEFINE VARIABLE cSourceType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile AS CHARACTER NO-UNDO.
DEFINE VARIABLE lOverrideDefaultMapping AS LOGICAL NO-UNDO.
DEFINE VARIABLE cFieldTypeMapping AS CHARACTER NO-UNDO.
DEFINE VARIABLE cVerifySchemaMode AS CHARACTER NO-UNDO.

ASSIGN
    cSourceType = "file"
    cFile = "ttcust.xsd"
    lOverrideDefaultMapping = NO
    cFieldTypeMapping = "address2,CLOB"
    cVerifySchemaMode = ?.

retOK = hTable:READ-XMLSCHEMA (cSourceType,
                               cFile,
                               lOverrideDefaultMapping,
                               cFieldTypeMapping,
                               cVerifySchemaMode).
```

Notes

- If the ProDataSet or temp-table object does not have a schema (that is, the object is dynamic and in the CLEAR state), Progress creates the schema from the XML Schema defined in the XML Schema document.

  If the ProDataSet or temp-table object already has a schema (that is, the object is static, or the temp-tables are in the PREPARED state), Progress verifies the XML Schema defined in the XML Schema document against the object's schema.

  If a dynamic temp-table is not in the PREPARED or CLEAR state, the method generates an error and returns FALSE.

  For more information about creating schema from XML Schema or verifying XML Schema, see *OpenEdge Development: Programming Interfaces*.

- You cannot create schema for a temp-table buffer or a database buffer.

See also

IS-XML attribute, READ-XML( ) method, WEB-CONTEXT system handle, WRITE-XML( ) method, WRITE-XMLSCHEMA( ) method, X-document object handle, X-noderef object handle

RECID attribute

The unique internal identifier of the database record currently associated with the buffer.

Data type: RECID

Access: Readable

Applies to: Buffer object handle

Note: Progress provides the RECID attribute (and the corresponding RECID function) for backward compatibility only. Progress Software Corporation recommends that in new code, you use the ROWID attribute.

RECORD-LENGTH attribute

The length, in bytes, of the record associated with a buffer.

Data type: LOGICAL

Access: Readable

Applies to: Buffer object handle

REFRESH( ) method

Forces Progress to refresh the display of the current rows in a browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
REFRESH ( )
```

If Progress successfully refreshes the widget, the method returns the value TRUE.

REFRESHABLE attribute

Indicates whether the rows that appear in a browse are refreshed when an application opens or repositions a query.

Note: When an application opens a query or repositions it multiple times, and refreshes the viewport each time, the display might flash, which is distracting. You can suppress the refreshing, and so reduce the flashing, by setting REFRESHABLE to FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

If REFRESHABLE is FALSE, when an application opens or repositions a query, the viewport is not refreshed. REFRESHABLE's default value is TRUE.

REFRESH-AUDIT-POLICY( ) method

Notifies the specified audit-enabled database that its audit policy tables have changed, which causes the database to refresh its current run-time audit policy settings cache before performing any more database operations.

For information about audit-enabling a database, or creating and activating an audit policy for a database, see *OpenEdge Getting Started: Core Business Services*.

Return type: LOGICAL

Applies to: AUDIT-POLICY system handle

Syntax

```
REFRESH-AUDIT-POLICY( integer-expression | logical-name | alias )
```

integer-expression

The sequence number of the connected database that needs to refresh its audit policy settings cache. For example, REFRESH-AUDIT-POLICY(1) notifies the first database, REFRESH-AUDIT-POLICY(2) notifies the second database, and so on. If you specify a sequence number that does not correspond to a connected database, Progress generates a run-time error.

logical-name or *alias*

The logical name or alias of the connected database that needs to refresh its audit policy settings cache. These forms require a quoted character string or a character expression. If you specify a logical name or alias that does not correspond to a connected database, Progress generates a run-time error.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

REGISTER-DOMAIN( ) method

Registers an authentication domain in the application's trusted authentication domain registry. Progress uses this registry to validate Client-principal objects during the session. After you have registered all authentication domains for a session, you must restrict the registration of additional domains by calling the LOCK-REGISTRATION( ) method.

Caution: Use caution when registering authentication domains in the application's trusted authentication domain registry using the REGISTER-DOMAIN( ) and LOCK-REGISTRATION( ) methods. You can introduce the risk for a security breach by allowing the registration of rogue domains between registering your domains and locking the registry. Consider using the LOAD-DOMAINS( ) method, which loads authentication domain registry information directly from an OpenEdge database and then automatically locks the registry.

Return type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
REGISTER-DOMAIN( domain-name, domain-key [, domain-description
  [, domain-type ] ] )
```

domain-name

A character expression that specifies the name of this authentication domain.

domain-key

A character expression that specifies the key to use when validating a Client-principal object created in this domain. Progress converts this key to UTF-8 before using it, which ensures a consistent value regardless of code page settings.

domain-description

An optional character expression that specifies a description for this domain.

domain-type

An optional character expression that specifies an application-defined domain type.

You must call the LOCK-REGISTRATION( ) method before you can use the trusted authentication domain registry to validate Client-principal objects for the application.

If you do not register at least one authentication domain in the trusted authentication domain registry before calling this method, this method returns TRUE. However, any attempt to seal a Client-principal object will generate a run-time error.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

Example The following code fragment illustrates how to use the REGISTER-DOMAIN( ) method:

```
DEF VAR name as CHAR.
DEF VAR key as CHAR.

FOR EACH trusted-registrar:
  name = trusted-registrar.name.
  key = trusted-registrar.key.
  SECURITY-POLICY:REGISTER-DOMAIN(name, key).
END.

SECURITY-POLICY:LOCK-REGISTRATION.
```

See also LOAD-DOMAINS( ) method, LOCK-REGISTRATION( ) method

REJECT-CHANGES( ) method

Rejects changes to the data in one temp-table or all temp-tables in a ProDataSet object.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle

Syntax

```
handle:REJECT-CHANGES( )
```

handle

A handle to the temp-table buffer or the ProDataSet object.

When you reject changes on a ProDataSet object handle, Progress uses the before-image table to back out changes from the after-image table, and empties the before-image table for each table in the ProDataSet.

When you reject changes for a Buffer object handle, Progress uses the before-image table to back out changes from the after-image table, and empties the before-image table for that one table.

As Progress rejects changes, it sets the BEFORE-ROWID attribute of every row in the after-image table to the Unknown value (?), and sets the ROW-STATE of every row in the after-image table to ROW-UNMODIFIED (0).

REJECT-ROW-CHANGES( ) method

Rejects changes to the data in one row of a ProDataSet temp-table.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
handle:REJECT-ROW-CHANGES( )
```

handle

A handle to a before-image temp-table buffer.

When you reject changes for a temp-table row, Progress uses the before-image table row to back out changes in the after-image table row. Progress also sets the BEFORE-ROWID attribute of the row in the after-image table to the Unknown value (?), sets the ROW-STATE of the row in the after-image table to ROW-UNMODIFIED (0), and removes the before-image table row.

REJECTED attribute

Set this attribute to indicate whether a change to the data in a ProDataSet object, a temp-table buffer, or a temp-table row is rejected.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

The REJECTED attribute corresponds to the REJECTED function.

The MERGE-CHANGES( ) method and MERGE-ROW-CHANGES( ) method use this attribute to determine whether to reject a changed row during a merge operation.

This attribute is marshalled between the client and the AppServer.

RELATION-FIELDS attribute

Returns the list of join fields between the parent and child as specified in the data-relation object definition.

Data type: CHARACTER

Access: Readable

Applies to: Data-relation object handle

You can use the value of this attribute in writing code that uses or extends the list of join fields without having to parse the value of the WHERE-STRING attribute, which in the default case provides essentially the same information but not necessarily in an ideal form for analyzing the relation.

RELATIONS-ACTIVE attribute

Indicates whether all data-relation objects in a ProDataSet object are active or inactive. Set to TRUE to activate all data-relation objects. Set to FALSE to deactivate all data-relation objects. All data-relation objects in a ProDataSet object are active by default.

Alternatively, you can activate or deactivate an individual data-relation object in a ProDataSet object by setting the ACTIVE attribute on the Data-relation object handle.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: ProDataSet object handle

Deactivate all data-relations in a ProDataSet object when you want a FILL operation to load data into each ProDataSet member buffer using the individual buffer's query (instead of recursively loading parent and child buffers as defined by the data-relations). Likewise, you can reactivate all data-relations in a ProDataSet object after completing a FILL operation to use the data-relations for traversing the data after the data is loaded.

When Progress encounters an inactive relation (or the last child buffer in the relation tree), during a FILL operation on a ProDataSet buffer object handle, Progress does not fill the child buffers of that relation. When Progress encounters an inactive relation during a FILL operation on a ProDataSet object handle, it treats the first child buffer of the inactive relation as a top-level table (including all records from its data source) and fills each child buffer based on the data relation's query. If you do not want Progress to treat the first child buffer of the inactive relation as a top-level table, set the FILL-MODE of that buffer to NO-FILL. Progress does not fill any of the child buffers.

If Progress encounters an inactive relation while navigating a ProDataSet object, it does not prepare or open a dynamic query for the child table, even if there is a browse associated with the relation's query. If you want to access the child temp-table, you must do so through a separate query, a FOR EACH statement, or some other standard Progress construct in your application code.

When you reactivate data-relations, Progress does not automatically resynchronize the hierarchy of queries on buffers below the newly active relation. If you want to resynchronize the related buffers, use the SYNCHRONIZE( ) method on the parent buffer.

REMOTE attribute

Indicates whether the specified procedure is running at the top level of an AppServer session as the result of a remote procedure call from a client application, or whether the current OpenEdge session is an AppServer session. Returns FALSE for a Web service procedure.

Data type: LOGICAL

Access: Readable

Applies to: SESSION system handle, THIS-PROCEDURE system handle (and all procedure handles)

For any procedure handle (including the THIS-PROCEDURE system handle), REMOTE is TRUE if:

- The specified procedure is running locally at the top level of the current session.
- The current session is an AppServer session.
- The procedure is running directly as the result of a remote procedure call from a client application.

Otherwise, REMOTE is FALSE. Thus, if the procedure handle is a proxy handle (PROXY attribute set to TRUE) or the specified procedure is running as the direct result of a call from any other procedure running in the current session context, REMOTE is FALSE.

For the SESSION handle, REMOTE is TRUE if the session runs in the context of an AppServer, and FALSE if the session runs in the context of a 4GL client.

For more information on the AppServer, see *OpenEdge Application Server: Developing AppServer Applications*.

REMOTE-HOST attribute

Indicates the IP (Internet Protocol) address of the machine the socket object is communicating with.

Data type: CHARACTER

Access: Readable

Applies to: Socket object handle

When a server and client successfully establish a connection, both the server and client have a socket object that identifies this connection. On the client, this attribute returns the IP address of the server, and on the server, this attribute returns the IP address of the client. If the CONNECT method fails or has not been called, this attribute returns the Unknown value (?).

REMOTE-PORT attribute

Indicates the port number of the socket.

Data type: INTEGER

Access: Readable

Applies to: Socket object handle

When a server and client successfully establish a connection, both the server and client have a socket object that identifies this connection. On the client, this attribute returns the port number used on the server machine for this socket connection. On the server, this attribute returns the port number used on the client machine for this socket connection. If the CONNECT failed, this attribute returns the Unknown value (?).

REMOVE-ATTRIBUTE( ) method

Removes the specified attribute of an element. If the removed attribute has a default value (specified by the document's DTD) it is set to its default value.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
REMOVE-ATTRIBUTE( name )
```

name

The name of the attribute to be removed.

The following example removes the attribute "Id", or resets the attribute "Id" to its default value, for the XML node represented by hNoderef:

```
hNoderef:REMOVE-ATTRIBUTE("Id").
```

REMOVE-CHILD( ) method

Unlinks the node and its sub-tree from the XML document. The XML object is not deleted, only disconnected from the structure.

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
REMOVE-CHILD( x-node-handle )
```

x-node-handle

The handle that represents the node to remove from the tree.

The following code fragment gets a reference to the fourth node on the document root, and removes it. hNoderef is still available for use after the remove, but is unlinked from hRoot:

```
CREATE X-NODEREF hNoderef.
...
hRoot:GET-CHILD(hNoderef,4).
hRoot:REMOVE-CHILD(hNodeRef).
. . .
```

REMOVE-EVENTS-PROCEDURE( ) method (Windows only; Graphical interfaces only)

Removes an external procedure from the list that Progress searches for event procedures to handle an ActiveX control event.

Return type: LOGICAL

Applies to: CONTROL-FRAME widget

Syntax

```
REMOVE-EVENTS-PROCEDURE ( procedure-handle )
```

procedure-handle

A handle to a persistent procedure or an otherwise active procedure on the call stack.

After removing the handle to an external procedure containing an event handler, the equivalent event procedure found in the next procedure in the search list handles the event. For information on this search list, see the ADD-EVENTS-PROCEDURE( ) method reference entry.

If the method succeeds in removing the procedure file from the list, it returns TRUE. Otherwise, it returns FALSE.

REMOVE-SUPER-PROCEDURE( ) method

Dissociates a super procedure file from a procedure file or from the current OpenEdge session. Returns FALSE for a Web service procedure.

Note: Dissociating a super procedure file from the current OpenEdge session does not automatically dissociate the super procedure file from procedure files within the session.

For more information on super procedures, see *OpenEdge Development: Progress 4GL Handbook*.

Return type: LOGICAL

Applies to: SESSION system handle, THIS-PROCEDURE system handle (and all procedure handles)

Syntax

```
REMOVE-SUPER-PROCEDURE ( super-proc-hdl  )
```

super-proc-hdl

A handle to the super procedure.

Note: If *super-proc-hdl* is not a valid procedure handle or is not currently a super procedure of the local procedure or of the current OpenEdge session, Progress does not report a run-time error.

REMOVE-SUPER-PROCEDURE returns FALSE if *super-proc-hdl* is not a valid handle. Otherwise, it returns TRUE.

The following code fragment dissociates a super procedure from the current procedure:

```
THIS-PROCEDURE:REMOVE-SUPER-PROCEDURE(my-super-proc-hdl).
```

The following code fragment dissociates a super procedure from a procedure file other than the current procedure:

```
local-proc-hdl:REMOVE-SUPER-PROCEDURE(my-super-proc-hdl).
```

The following code fragment dissociates a super procedure from the current OpenEdge session:

```
SESSION:REMOVE-SUPER-PROCEDURE(my-super-proc-hdl).
```

REPLACE( ) method

Replaces an item in a combo box, radio set, or selection list. Replaces an existing text string in an editor with a new text string.

Return type: LOGICAL

Applies to: COMBO-BOX widget, EDITOR widget, RADIO-SET widget, SELECTION-LIST widget

Combo-box and Selection-list syntax

```
REPLACE (
  { new-item-list | new-label , new-value }
  ,
  { list-item | list-index } )
```

new-item-list

A character-string expression that specifies a single item or a delimiter-separated list of items to add to the widget.

new-label

A character-string expression that specifies the label of a label-value pair to add to the widget.

new-value

The new value assigned when a user selects the label.

Note: Use *new-item-list* when the widget's entries consist of single items. Use *new-label* and *new-value* when the widget's entries consist of label-value pairs.

list-item

A character-string expression that specifies a single value in the widget.

list-index

An INTEGER expression that specifies the ordinal position of an existing entry in the combo box list or selection list.

For combo boxes and selection lists, REPLACE replaces *list-item* with one of the following: *new-label-list*, or the label-value pair represented by *new-label* and *new-value*. If *list-item* is currently selected, the new item is not selected when it appears in the list. If the method is successful, it returns TRUE.

Editor syntax:

```
REPLACE ( old-string , new-string , flag )
```

old-string

A character-string expression to be replaced. For the large editor widget in Windows, you can use wildcard characters for regular expression pattern matching. A question mark(?) in a particular position indicated that any single character is acceptable in that position. An asterisk (*) indicates that any group of characters is acceptable, including a null group of characters. If you want to specify a question mark (?) or asterisk (*) as a literal character rather than a wildcard character in the string, use ?? and ** respectively.

new-string

A character-string expression to replace *old-string*.

flag

An integer expression that specifies the type of search to be performed.

For editors, REPLACE searches from the current text cursor position for an occurrence of *old-string* and replaces it with *new-string*. If the replace operation is successful, the method returns TRUE. The *flag* value determines the type of search and replace to perform. Table 81 lists the flag values that correspond to each search and replace type.

Table 81: REPLACE flag values

Type of search	Flag value
FIND-NEXT-OCCURRENCE	1
FIND-PREV-OCCURRENCE	2
FIND-CASE-SENSITIVE	4
FIND-GLOBAL	8
FIND-WRAP-AROUND	16

For a single operation, you cannot specify both FIND-NEXT-OCCURRENCE and FIND-PREV-OCCURRENCE, nor can you specify both FIND-WRAP-AROUND and FIND-GLOBAL. All other combinations of flags are valid. For example, you can specify a combination of FIND-NEXT-OCCURRENCE + FIND-GLOBAL + FIND-CASE-SENSITIVE. The default is FIND-NEXT_OCCURRENCE to search to the end of the editor string.

Radio-set syntax

```
REPLACE ( new-label , new-value , old-label )
```

new-label

A character-string expression that specifies the new item label.

new-value

The new value assigned when a user selects the item.

old-label

A character-string expression that specifies the label of the item to be replaced.

The REPLACE( ) method for radio sets replaces the label, the value, or both the label and value of the specified radio item. To retain the existing label or value, substitute an empty string.

REPLACE(*new-label* , *new-value* , *old-label*) replaces the specified radio item with a new item, consisting of both a new label and a new value.

REPLACE(*new-label* , *""* , *old-label*) replaces only the label of the specified radio item, retaining the value.

REPLACE(*""* , *new-value* , *old-label*) replaces only the value of the specified radio item, retaining the label.

If the new label is longer than the existing radio set size can accommodate, the radio set appearance changes depending on setting of the AUTO-RESIZE attribute. If AUTO-RESIZE is TRUE, the radio set expands to accommodate the label. If AUTO-RESIZE is FALSE, the new label is clipped to fit the current size. However, note that the label is clipped only on the display. The new radio set item is identified by the full label regardless of its length.

If the replace operation is successful, the method returns TRUE.

REPLACE-CHILD( ) method

Replace an old XML node with a new node. The old XML node is not deleted, only disconnected from the structure. If the new XML node is already in the tree, it is first disconnected.

Return type: LOGICAL

Applies to: X-document object handle, X-noderef object handle

Syntax

```
REPLACE-CHILD( new-handle , old-handle )
```

new-handle

The handle that represents the node to insert in the tree.

old-handle

The handle that represents the node to remove from the tree.

The following code fragment gets a reference to the fourth XML node on the document root, and removes it. hNoderef is still available for use after the remove, but is unlinked from hRoot. We then replace the root's second child with this fourth child:

```
CREATE X-NODEREF hNoderef.
CREATE X-NODEREF hChild.
...
hRoot:GET-CHILD(hNoderef,4).
hRoot:REMOVE-CHILD(hNodeRef).
hRoot:GET-CHILD(hChild,2).
hRoot:REPLACE-CHILD(hNodeRef,hChild).
```

REPLACE-SELECTION-TEXT( ) method

Replaces the currently selected text in an editor widget with the new text.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
REPLACE-SELECTION-TEXT ( new-text )
```

new-text

A character-string expression that specifies the new text to replace the currently selected text.

To determine what text is currently selected, query the SELECTION-TEXT attribute. If the replace operation is successful, the method returns TRUE.

REPOSITION attribute

The reposition mode of a data-relation object. If TRUE, the relation mode is REPOSITION. If FALSE, the relation mode is SELECTION. The default value is FALSE.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Data-relation object handle

REPOSITION-BACKWARD( ) method

Moves a query object's result list pointer back a particular number of rows.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
REPOSITION-BACKWARD ( n )
```

n

An INTEGER expression representing the number of rows.

REPOSITION-BACKWARD( ) always places the cursor between rows. For example:

- If the cursor in on a row—say, row 5—REPOSITION-BACKWARD(1) moves the cursor to row 4, then to half way between rows 4 and 5. From this position, GET-PREV( ) moves the cursor to row 4, while GET-NEXT( ) moves the cursor to row 5.
- If the cursor is between two rows—say, between rows 5 and 6—REPOSITION-BACKWARD(1) moves the cursor to half way between rows 4 and 5. From this position, GET-PREV( ) moves the cursor to row 4, while GET-NEXT( ) moves the cursor to row 5.

Note: The REPOSITION-BACKWARD method corresponds to the REPOSITION statement BACKWARDS option.

REPOSITION-FORWARD( ) method

Moves a query object's result list pointer forward a particular number of rows.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
REPOSITION-FORWARD ( n )
```

n

An INTEGER expression representing the number of rows.

REPOSITION-FORWARD( ) always places the cursor between rows. For example:

- If the cursor in on a row—say, row 5—REPOSITION-FORWARD(1) moves the cursor to row 6, then to half way between rows 6 and 7. From this position, GET-PREV( ) moves the cursor to row 6, while GET-NEXT( ) moves the cursor to row 7.
- If the cursor is between two rows—say, between rows 5 and 6—REPOSITION-FORWARD(1) moves the cursor to half way between rows 6 and 7. From this position, GET-PREV( ) moves the cursor to row 6, while GET-NEXT( ) moves the cursor to row 7.

Note: The REPOSITION-FORWARD method corresponds to the REPOSITION statement FORWARDS option.

REPOSITION-TO-ROW( ) method

Moves a query object's result list pointer to the row corresponding to the specified sequence number.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
REPOSITION-TO-ROW ( n )
```

n

An INTEGER expression representing the sequence number.

Note: The REPOSITION-TO-ROW method corresponds to the REPOSITION statement TO ROW option.

REPOSITION-TO-ROWID( ) method

Moves a query object's result list pointer to the row corresponding to the specified ROWID or ROWIDs.

To reposition to a particular row when the query is a join, supply the ROWIDs of the buffers that correspond to the desired row.

Return type: LOGICAL

Applies to: Query object handle

Syntax

```
REPOSITION-TO-ROWID (
  {     rowid1 [ , rowid2 ] ...
    |   rowid-array
  }
)
```

rowid1 **[** , *rowid2* **]** **...**

Expressions of type ROWID representing the rowid of the first buffer, the rowid of the second buffer, etc. The maximum number of expressions is 18. If an expression contains the Unknown value (?), Progress evaluates but ignores subsequent expressions.

rowid-array

An array of 18 or fewer elements, where each element is of type ROWID and represents the rowid of a buffer. If an element contains the Unknown value (?), Progress evaluates but ignores subsequent elements.

Note: The REPOSITION-TO-ROWID method corresponds to the REPOSITION statement TO ROWID option.

RESET( ) method

Closes the open stream and resets the SAX-writer object to its default values.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
RESET( )
```

Use this method to reuse a SAX-writer object for multiple documents or to cancel a write.

The object attributes and the output destination remain unchanged. The WRITE-STATUS attribute is set to SAX-WRITE-IDLE.

You can call this method at any time. If the WRITE-STATUS attribute is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE, the method call has no effect. Otherwise, the method call closes the document and stream, and aborts the write.

See also WRITE-STATUS attribute

RESIZABLE attribute (Graphical interfaces only)

Indicates whether the user can resize a widget at run time.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

If you set RESIZABLE to TRUE, the user can resize the widget. For the RESIZABLE attribute to take effect, you must also set the SELECTABLE attribute to TRUE.

Note: Setting the RESIZABLE attribute to TRUE enables direct manipulation events for the widget. These events take precedence over all other events. This effectively prevents data entry using the widget until all direct manipulation events are disabled (that is, until MOVABLE, RESIZABLE, and SELECTABLE are all FALSE).

RESIZE attribute (Graphical interfaces only)

Indicates whether the user can resize a window at run time.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

If the RESIZE attribute is TRUE, users can resize the window.

You can set this attribute only before the window is realized.

RESTART-ROWID attribute

Specifies the ROWID of the data source row at which a FILL operation will start. Set this attribute before each FILL operation in a series of FILL operations to retrieve data source rows in batches. You typically set this attribute by assigning the value of the NEXT-ROWID attribute, which is set by Progress after each FILL operation.

Note: The NEXT-ROWID attribute is not marshalled between the client and the AppServer. You are responsible for retrieving, storing, and transporting this attribute value between the client and the AppServer.

Data type: ROWID

Access: Readable/Writeable

Applies to: Data-source object handle

Syntax

```
data-source-handle:RESTART-ROWID( buffer-sequence-number | buffer-name )
```

data-source-handle

The handle to the Data-source object.

buffer-sequence-number

An INTEGER that represents the sequence number of a buffer in the list of buffers for the Data-source object. Specify *buffer-sequence-number* to identify a buffer in the Data-source object when the Data-source object is defined against more than one database table buffer. The default is the first (or only) buffer in the Data-source object.

Note: Sequence numbers for buffers in a Data-source object start at one, where one represents the top level and subsequent numbers represent lower levels of join, if any.

buffer-name

A CHARACTER expression that evaluates to the name of a buffer in the list of buffers for the Data-source object.

If an invalid buffer is specified, this attribute returns the Unknown value (?).

When specified, the next FILL operation using this data source opens its associated query and tries to reposition the query to the given ROWID. If the reposition is successful, Progress proceeds with the FILL operation. If the reposition is not successful, Progress sets the RESTART-ROWID attribute to the Unknown value (?) and proceeds with the FILL operation from the top of the query at the current level (that is, without repositioning the query).

This attribute is not marshalled between the client and the AppServer.

See also BATCH-SIZE attribute, FILL( ) method, NEXT-ROWID attribute

RETAIN-SHAPE attribute

Indicates that the image should retain its aspect ratio (expand or contract equally in both dimensions).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: IMAGE widget

Setting RETAIN-SHAPE to TRUE may leave some uncovered space at the bottom or right of the image widget. RETAIN-SHAPE is ignored if STRETCH-TO-FIT is equal to FALSE or if an icon is displayed on the image widget.

RETURN-INSERTED attribute (Windows only)

How an editor widget behaves when a RETURN event occurs.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

If the RETURN-INSERTED attribute is TRUE, a RETURN event inserts a hard return at the cursor position, breaking the current line. Otherwise, if the editor is in a dialog box with a default button, a RETURN event chooses the default button for the dialog box. If the editor is not in a dialog box with a default button, a RETURN event inserts a hard return for any value of RETURN-INSERTED. The default value is FALSE.

You can set this attribute only before the editor widget is realized.

RETURN-VALUE attribute

The value returned from user-defined functions, attributes, and methods.

Note: Supported only if the CALL-TYPE attribute is set to FUNCTION-CALL-TYPE or GET-ATTR-CALL-TYPE.

Data type: The value of the RETURN-VALUE-DATA-TYPE attribute.

Access: Readable

Applies to: CALL object handle

The default is the Unknown value (?).

If you set the RETURN-VALUE-DATA-TYPE attribute to a particular data type before executing INVOKE( ), the value returned is converted to that data type.

If RETURN-VALUE-DATA-TYPE is set to its default value or to the expected data type, when the dynamic invoke returns, the value returned by the user-defined function or attribute is not converted.

RETURN-VALUE-DATA-TYPE attribute

When you dynamically invoke a user-defined function, get an attribute, or run a method, RETURN-VALUE-DATA-TYPE lets you do one of the following to the return value:

- Detect its data type.
- Cause it to be converted to a different data type.

If you set RETURN-VALUE-DATA-TYPE to a data type other than the expected data type before executing the dynamic invoke, the value returned is automatically converted to that data type. Otherwise, when the dynamic invoke returns, RETURN-VALUE-DATA-TYPE indicates the data type of the value returned.

Data type: CHARACTER

Access: Readable/Writable

Applies to: CALL object handle

Syntax

```
RETURN-VALUE-DATA-TYPE [ = datatype ]
```

datatype

A CHARACTER expression indicating one of the following:

- "CHARACTER"
- "DATE"
- "LOGICAL"
- "INTEGER"
- "DECIMAL"
- "RAW"
- "HANDLE"

The default is the Unknown value (?).

ROLES attribute

Returns a comma-separated list of authentication domain roles for the user ID associated with the Client-principal object. This list cannot contain embedded spaces. If not specified, Progress returns a zero-length character string.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

Once the Client-principal object is sealed, this attribute is read-only.

ROW attribute

The row position of the top edge of the widget relative to the top edge of the current iteration of a parent widget or the display. Specifies the row position of the mouse cursor for the last mouse event relative to the top edge of the display.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget, LAST-EVENT system handle

This attribute is read-only for browse columns, field groups, and the LAST-EVENT handle.

For all widgets except windows, the ROW attribute specifies the location, in character units, of the top edge of the widget relative to the top edge of its parent widget. In windows, the location is relative to the top edge of the display.

For a browse column, the ROW attribute returns the Unknown value (?) if the column is hidden.

If the parent is a down frame with multiple occurrences, the ROW attribute regards the parent as the current occurrence.

For control-frames, the ROW attribute maps to the Top property of the control-frame COM object (ActiveX control container).

For the LAST-EVENT handle, the ROW attribute specifies the row location, in character units, of the last mouse event relative to the top edge of the current frame.

This attribute is functionally equivalent to the Y attribute.

ROW-HEIGHT-CHARS attribute (Graphical interfaces only)

Sets the row height, in characters, of a browse.

Data type: DECIMAL

Access: Readable/Writable

Applies to: BROWSE widget

Note: In browses, all rows have the same height.

ROW-HEIGHT-PIXELS attribute (Graphical interfaces only)

Sets the row height, in pixels, of a browse.

Data type: INTEGER

Access: Readable/Writable

Applies to: BROWSE widget

Note: In browses, all rows have the same height.

ROW-STATE attribute

The current change state of the ProDataSet temp-table row associated with the buffer object handle.

Data type: INTEGER

Access: Readable

Applies to: Buffer object handle

Note: The ROW-STATE attribute corresponds to the ROW-STATE function.

When the TRACKING-CHANGES attribute is set to TRUE for a ProDataSet temp-table, Progress tracks changes to the data in that temp-table using a before-image temp-table that contains the original version of each row. You can think of the temp-table itself as the after-image because it contains the latest version of each row.

Every row in the after-image table that has been modified or created corresponds to a row in the before-image table. Deleted rows do not appear in the after-image table, because it reflects the current state of the data. Every row in the before-image table has a non-zero ROW-STATE, because every row is the before-image of a deleted, created, or modified row in the after-image table. Unchanged rows do not appear in the before-image table.

You can use the ROW-STATE attribute on each row in either the after-image table or the before-image table to determine whether a row has changed and how it has changed.

The possible values can be expressed as compiler constants. Table 82 lists these values.

Table 82: Row state values

Compiler constant	Value	Description
ROW-UNMODIFIED	0	The row was not modified.
ROW-DELETED	1	The row was deleted.
ROW-MODIFIED	2	The row was modified.
ROW-CREATED	3	The row was created.

ROWID attribute

The unique internal identifier of the database record currently associated with the buffer.

Data type: ROWID

Access: Readable

Applies to: Buffer object handle

Note: The ROWID attribute corresponds to the ROWID function.

ROW-MARKERS attribute

Indicates whether a browse uses row markers.

Data type: LOGICAL

Access: Readable

Applies to: BROWSE widget

If this attribute is set to TRUE, the browse has row markers.

ROW-RESIZABLE attribute (Graphical interfaces only)

Indicates whether you can change a browse's row height.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget

If ROW-RESIZABLE is TRUE, Progress sensitizes the browse to the START-ROW-RESIZE and END-ROW-RESIZE events, which lets the user change the row height.

Note: In a browse, all rows have the same height.

SAVE( ) method

Saves or sends an XML document.

Return type: LOGICAL

Applies to: X-document object handle

Syntax

```
SAVE( mode , { file | stream | memptr | longchar } )
```

mode

A character expression that evaluates to one of the following: “FILE”, “STREAM”, “MEMPTR”, or “LONGCHAR”. The *mode* indicates whether the XML target is a file, a stream, a MEMPTR, or a LONGCHAR variable.

file

A character expression that represents the name of a new file to be created in current working directory of the underlying operating system’s file system. If a file with the specified name already exists, Progress verifies that the file is writeable and overwrites the file.

stream

A character expression that represents the name of a 4GL stream. If *stream* is "", Progress saves the document to the 4GL session unnamed stream.

memptr

A MEMPTR variable to contain the saved XML text in memory. The SAVE method allocates the required amount of memory and sets the size of the variable. You must release the memory later with a SET-SIZE( ) = 0.

longchar

A LONGCHAR variable to contain the saved XML text in memory.

Progress saves the LONGCHAR variable in the code page of the XML document, as determined by the XML document's ENCODING attribute. If the XML document's ENCODING attribute has not been set, the LONGCHAR variable is saved in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE function) and the fixed code page is not equivalent to the encoding specified in the XML document's ENCODING attribute, the SAVE( ) method returns an error and the XML document is not saved to the LONGCHAR.

Example The following code fragments demonstrate the use of the SAVE( ) method:

```
/* Saves the current tree under hDoc as memo.xml. */
DEFINE STREAM mystream.
OUTPUT STREAM mystream to memo.xml.
hdoc:SAVE("stream","mystream").

OR

/* Saves the current tree under hDoc as memo.xml. */
hdoc:SAVE("file","memo.xml").

OR

/* Saves the current tree under hDoc in memory referred to by mymem. */
DEFINE VARIABLE hDoc AS HANDLE.
DEFINE VARIABLE mymem AS MEMPTR.
. . .
hDoc:SAVE("memptr", mymem).
```

See also ENCODING attribute

SAVE-FILE( ) method

Saves the current contents of the editor widget to a specified text file and sets the widget's MODIFIED attribute to FALSE.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
SAVE-FILE ( filename )
```

filename

A character-string expression of a full or relative pathname of a file.

If the save is not successful, it does not change the value of the MODIFIED attribute. If the save is successful, the method returns TRUE.

In Windows, this method writes out text files with a carriage return character and a line feed character terminating each line of text. In all other interfaces, this method writes out text files with a carriage return character terminating each line of text. Also in Windows, **RETURN** key input writes out as x0d0d0a and in all other interfaces, writes out as x0d0a.

SAVE-ROW-CHANGES( ) method

Saves changes from one row of a ProDataSet temp-table to the associated data source.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
handle:SAVE-ROW-CHANGES( [ buffer-index | buffer-name
                         [ , skip-list [ , no-lobs ] ] ] )
```

handle

A handle to a before-image buffer in a ProDataSet object.

buffer-index

An INTEGER expression that specifies the index of the buffer in the data source's buffer list. The default value is 1.

buffer-name

A CHARACTER expression that evaluates to the name of the buffer in the data source.

skip-list

An optional character expression that evaluates to a comma-separated list of field names for fields that should not be assigned after a new row is created (that is, fields to skip). For example, a key field or other fields assigned a value by a CREATE database trigger.

no-lobs

A logical expression indicating whether to ignore BLOB and CLOB fields in the save operation. If TRUE, BLOB and CLOB fields are ignored during the save operation. If FALSE, BLOB and CLOB fields are saved along with the other fields. The default value is FALSE (that is, BLOB and CLOB fields are included in the save operation).

The data source must be attached before calling this method. If there is no data source, or special processing is needed, you must write the code to save the changes instead of using the SAVE-ROW-CHANGES( ) method.

Progress saves ProDataSet buffer changes to the associated data source based on the current MERGE-BY-FIELD attribute and PREFER-DATASET attribute settings.

When MERGE-BY-FIELD is TRUE, Progress merges changes from a ProDataSet temp-table buffer to the associated data source on a field-by-field basis. When MERGE-BY-FIELD is FALSE, Progress does not merge changes on a field-by-field basis.

Note: Merging a large number of changes from a ProDataSet object to the data source on a field-by-field basis is slower than saving changes buffer-by-buffer.

Before saving any changes, Progress compares the before-image of the ProDataSet temp-table buffer or field, saved while tracking changes for the buffer, to the corresponding buffer or field in the data source to determine whether the data in the data source has changed since being read.

If the data in the data source has not changed, Progress copies the ProDataSet buffer or field to the data source. If the data in the ProDataSet buffer or field was deleted, Progress deletes the data from the data source.

If the data in the data source has changed, Progress saves the ProDataSet buffer changes based on the current PREFER-DATASET attribute setting. When PREFER-DATASET is TRUE, Progress copies the data from ProDataSet buffer or field to the data source regardless of any changes made to the data in the data source since the data was read. When PREFER-DATASET is FALSE, Progress does not copy the data from the ProDataSet buffer or field to the data source. Progress copies the conflicting data from the data source to the ProDataSet buffer or field, instead, and sets the ERROR attribute and DATA-SOURCE-MODIFIED attribute to TRUE.

If Progress encounters an error, it sets the value of the ERROR attribute to TRUE for the associated ProDataSet object handle, Temp-table object handle, and Buffer object handle.

SAVE-WHERE-STRING attribute

Returns the WHERE clause used to find the database buffer identified by the buffer-index or buffer-name in the Data-source object.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Data-source object handle

Syntax

```
data-source-handle:SAVE-WHERE-STRING( buffer-index | buffer-name )
```

data-source-handle

The handle to the Data-source object.

buffer-index

The 1-based index of the buffer in the list of buffers for the Data-source object.

buffer-name

The name of a before-image table in the list of buffers for the Data-source object. The default is the first (or only) buffer in the Data-Source buffer list.

SAX-PARSE( ) method

Performs a single-call parse of an XML document associated with a SAX-reader object.

Return type: None

Applies to: SAX-reader object handle

Syntax

```
SAX-PARSE ( )
```

Returns when one of the following occurs:

- The parser reaches the end of the document.
- A callback invokes RETURN ERROR.
- An error occurred because the parser could not start or could not continue parsing.

While SAX-PARSE( ) is running, each time the parser detects an XML element, Progress invokes the corresponding callback. Callbacks reside in a procedure file whose handle has been assigned to the HANDLER attribute.

Note: SAX-PARSE( ) does not have a return value. To detect if an error has occurred, add NO-ERROR to the call and when it returns, check ERROR-STATUS:ERROR.

SAX-PARSE-FIRST( ) method

Initializes and begins a progressive-scan parse of an XML document associated with a SAX-reader object.

Return type: None

Applies to: SAX-reader object handle

Syntax

```
SAX-PARSE-FIRST ( )
```

To continue a progressive-scan parse, call SAX-PARSE-NEXT( ) repeatedly.

When SAX-PARSE-FIRST( ) executes, Progress invokes the StartDocument callback, if there is one.

SAX-PARSE-FIRST( ) raises a Progress error if the parser fails, for any reason, to start parsing the XML source.

If SAX-PARSE-FIRST( ) is called on a source file already being parsed with SAX-PARSE-NEXT( ), the parser reinitializes the parsing process.

SAX-PARSE-FIRST( ) can be called with SAX-reader in any state.

Note: SAX-PARSE-FIRST( ) does not have a return value. To detect if an error has occurred, add NO-ERROR to the call and when it returns, check ERROR-STATUS:ERROR.

SAX-PARSE-NEXT( ) method

Continues a progressive-scan parse of an XML document associated with a SAX-reader object.

Return type: None

Applies to: SAX-reader object handle

Syntax

```
SAX-PARSE-NEXT ( )
```

SAX-PARSE-NEXT( ) returns after the parser finds the next XML token in the XML source and Progress invokes the corresponding callback, if it exists.

When SAX-PARSE-NEXT( ) starts, an error is raised if the PARSE-STATUS attribute is anything other than SAX-RUNNING. This might occur if:

- SAX-PARSE-NEXT( ) is called before SAX-PARSE-FIRST( ) is called.
- STOP-PARSING( ) is called.

Eventually, SAX-PARSE-NEXT( ) fails to find another XML token (assuming the parse does not stop early). When this happens, Progress sets the PARSE-STATUS attribute to SAX-COMPLETE. If SAX-PARSE-NEXT( ) is called at that point, an error is raised.

Note: SAX-PARSE-NEXT( ) does not have a return value. To detect if an error has occurred, add NO-ERROR to the call and when it returns, check ERROR-STATUS:ERROR.

To stop a progressive-scan parse, you can refraining from calling SAX-PARSE-NEXT( ). However, when you detect that it is time to stop, you will most likely be in a callback, so you need a way to communicate that it is time to stop to the driver (where SAX-PARSE-NEXT( ) is called). A convenient way to do that is to call SELF:STOP-PARSING( ), since that causes Progress to set the value of PARSE-STATUS to SAX-COMPLETE.

SCHEMA-CHANGE attribute

Relaxes the requirement for exclusive access to a database in order to make the following schema changes online:

- Add new sequences.
- Add new tables, as well as any associated fields, indexes, and database triggers (which must be added within the same transaction).
- Add new fields to an existing table. (You cannot define ASSIGN triggers for new fields while the database is online.)
- Add new inactive indexes to an existing table.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

Set this attribute to “NEW OBJECTS” to activate this feature; and to the empty string ("") or the Unknown value (?) to deactivate this feature.

SCHEMA-LOCATION attribute

Determines the list of namespace/location pairs of an XML Schema file to validate against.

Data type: CHARACTER

Access: Readable/Writable

Applies to: X-document object handle, SAX-reader object handle

Contains the XML Schema file namespace/location pairs list for the object that the parser is using to validate against. The value pairs are separated by white space.

The default is an empty string ("").

The syntax for the list is the same as the attribute `schemaLocation` in an XML document. The first member of each pair is the namespace and the second member is the location where to find to an appropriate XML Schema file. The list is written as pairs of target namespace and locations where each value is separated by white space, using the format below:

```
namespace1 location1 namespace2 location2 namespace3 location3
```

Here is an example of this method call:

```
hXdoc:SCHEMA-LOCATION = "http://www.example.com com.xsd
                         http://www.example.org org.xsd"
```

Note that namespace and XML Schema file locations specified programatically with this method take precedence over namespaces or schemas declared in XML documents or imported elements.

To clear the schema location list set SCHEMA-LOCATION to an empty string ("").

The above example could also be achieved using the ADD-SCHEMA-LOCATION( ) method twice:

```
hXdoc:ADD-SCHEMA-LOCATION("http://www.example.com", "com.xsd").
hXdoc:ADD-SCHEMA-LOCATION("http://www.example.org", "org.xsd").
```

SCHEMA-MARSHAL attribute

Specifies the amount of schema information to marshal for a temp-table parameter. The temp-table may be an independent temp-table or a member of a ProDataSet object. The default value is FULL (which includes all schema information for the temp-table parameter).

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Temp-table object handle

Table 83 lists the SCHEMA-MARSHAL attribute values.

Table 83: SCHEMA-MARSHAL attribute values

Attribute value	Description
FULL	Includes all schema information for a temp-table parameter.
MIN	Minimizes schema information for a temp-table parameter.
NONE	Excludes schema information for a temp-table parameter.

If set to the Unknown value (?), the default value is FULL.

Note: If you specify both the SCHEMA-MARSHAL attribute and the MIN-SCHEMA-MARSHAL or NO-SCHEMA-MARSHAL attribute for an individual temp-table, Progress uses the attribute you most recently specified. The MIN-SCHEMA-MARSHAL and NO-SCHEMA-MARSHAL attributes are supported only for backward compatibility. Use the SCHEMA-MARSHAL attribute instead.

Setting this attribute overrides the setting of the Temp-table Schema Marshal (`-ttmarshal`) startup parameter for an individual temp-table parameter. For more information about this startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

See also MIN-SCHEMA-MARSHAL attribute, NO-SCHEMA-MARSHAL attribute

SCHEMA-PATH attribute

A delimiter-separated list of directory paths used to locate the XML Document Type Definition (DTD) associated with a particular XML document. It is searched if the XML document contains a relative path to the DTD.

Data type: CHARACTER

Access: Readable/Writable

Applies to: SAX-reader object handle, X-document object handle

The default is the Unknown value (?).

To separate directory paths, use a comma, or a delimiter character that your operating system recognizes.

To separate directory names and filenames within a path, use the UNIX format (forward slashes) or the standard format for your operating system.

SCHEMA-PATH can include HTTP URLs, but they might slow your application since accessing a DTD over the Web might take significantly longer than doing so over a file network.

SCHEMA-PATH cannot contain Progress procedure libraries (`.pl` files).

File pathnames cannot contain embedded commas. Progress replaces delimiter characters with commas, so the resulting SCHEMA-PATH can be accessed using the ENTRY( ) function.

Note: When Progress searches for the DTD, Progress concatenates each entry in SCHEMA-PATH with SYSTEM-ID (an attribute in DOM and a parameter of the ResolveEntity callback in SAX), which indicates the system ID of the external DTD. SYSTEM-ID might be as simple as a filename or as complex as a relative path.

For more information on accessing XML documents using the Document Object Model (DOM) or Simple API for XML (SAX) interfaces, see *OpenEdge Development: Programming Interfaces*.

SCREEN-LINES attribute

The number of display lines available in the window, in character units.

Data type: DECIMAL

Access: Readable

Applies to: WINDOW widget

SCREEN-VALUE attribute

The data value in the screen buffer associated with the widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget (cell), COMBO-BOX widget, EDITOR widget, FILL-IN widget, LITERAL widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Note that setting the SCREEN-VALUE attribute does not affect the record buffer. To apply the updated value to the record buffer you must explicitly assign the field or variable. Likewise, assigning the record buffer does not affect the screen buffer. To display a value in the record buffer, you must explicitly assign it to the SCREEN-VALUE attribute or implicitly move it to the screen buffer using a DISPLAY or UPDATE statement.

For combo boxes, this attribute returns the screen buffer value of the combo box fill-in. If no item in the list is selected or the list is empty, this attribute returns the Unknown value (?). Setting this attribute to an item in the list deselects the previously selected item and assigns the value of the selected item to the fill-in screen buffer. For SIMPLE and DROP-DOWN combo boxes, if the new value in the fill-in is not an item in the list, the fill-in screen buffer is set to the new value and no item in the list is selected. For DROP-DOWN-LIST combo boxes, if the new value in the fill-in is not an item in the drop down list, Progress ignores the value and displays a warning message.

For browse cells, screen values are applied to the buffers automatically when the user leaves the row. If the browse has the NO-ASSIGN option specified in the DEFINE BROWSE statement or it is a dynamic browse, then you must apply the screen values.

Changing the SCREEN-VALUE attribute for a browse column is useful in setting the cells in a non-database browse column.

Note: If you assign a value to the SCREEN-VALUE attribute of a widget and display the widget using a DISPLAY or UPDATE statement, the value you assigned appears as the initial value.

If a selection list is not already realized and you reference its SCREEN-VALUE attribute, Progress realizes the widget.

In Windows, SCREEN-VALUE applies to the regular editor and to the large editor.

SCROLL-BARS attribute

Indicates whether scroll bars appear in a window.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

If the SCROLL-BARS attribute is TRUE, scroll bars appear for the window when a user resizes the window to a size smaller than virtual height or width of the window. The RESIZE attribute controls the ability to resize a window. The VIRTUAL-HEIGHT-CHARS, VIRTUAL-HEIGHT-CHARS, VIRTUAL-HEIGHT-CHARS, and VIRTUAL-HEIGHT-CHARS attributes control the virtual or maximum size of a window.

SCROLL-TO-CURRENT-ROW( ) method

Scrolls a browse (if necessary) to bring the currently selected row into view. If the browse supports multiple selections, then SCROLL-TO-CURRENT-ROW( ) brings the most recently selected row into view.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SCROLL-TO-CURRENT-ROW ( )
```

The position of the scrolled row is the first row in the browse viewport, unless the current row is already visible. In this case, the current row remains in the original position. If the row is successfully scrolled into view (or if the scroll is unnecessary), the method returns TRUE.

SCROLL-TO-ITEM( ) method

Scrolls a selection list so that the specified item appears at the top of the list.

Return type: LOGICAL

Applies to: SELECTION-LIST widget

Syntax

```
SCROLL-TO-ITEM ( list-item | list-index )
```

list-item

A character-string expression that specifies a single value in the selection list.

list-index

An integer expression that specifies the ordinal position (first, second, third, etc.) of an entry in the selection list.

If the method is successful, it returns TRUE.

SCROLL-TO-SELECTED-ROW( ) method

Scrolls a browse (if necessary) to bring a specified selected row into view.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SCROLL-TO-SELECTED-ROW ( n )
```

n

An integer expression that specifies a selected row within the browse.

Progress maintains a numbered list of selected rows, starting at 1. When the SCROLL-TO-SELECTED-ROW(*n*) method is encountered, Progress searches this list to find the *n*th selected row. If the row is successfully scrolled into view (or if the scroll is unnecessary), the method returns TRUE.

SCROLLABLE attribute

The scrolling capabilities of a frame or a dialog box.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget

If the SCROLLABLE attribute is TRUE, the frame or dialog box can be bigger than the display space allotted to it (that is, it is scrollable). If SCROLLABLE is FALSE, the frame or dialog box must fit within the allotted display space; it cannot be made to scroll. The default value is FALSE. The VIRTUAL-HEIGHT-CHARS, VIRTUAL-HEIGHT-CHARS, VIRTUAL-HEIGHT-CHARS, and VIRTUAL-HEIGHT-CHARS attributes control the virtual or maximum size of a frame or dialog box.

SCROLLBAR-HORIZONTAL attribute

Indicates whether a horizontal scroll bar appears in an editor or a selection list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget, SELECTION-LIST widget

If the SCROLLBAR-HORIZONTAL attribute is TRUE, a horizontal scroll bar appears on the bottom edge of the widget. For an editor widget, horizontal scrolling is always enabled whether or not a horizontal scroll bar is enabled. For a selection list, the scroll bar must be enabled to scroll the list. The default value is FALSE.

Note: If the SCROLLBAR-HORIZONTAL attribute is set to TRUE, then WORD-WRAP is automatically set to FALSE. Likewise, if you set the WORD-WRAP attribute to TRUE, then SCROLLBAR-HORIZONTAL is automatically set to FALSE.

You can set this attribute only before the widget is realized.

SCROLLBAR-VERTICAL attribute (Graphical interfaces only)

Indicates whether a vertical scroll bar appears in a browse, editor or a selection list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, EDITOR widget, SELECTION-LIST widget

If the SCROLLBAR-VERTICAL attribute is TRUE, a vertical scroll bar appears on the right side of the widget. For an editor widget, vertical scrolling is always enabled whether or not a vertical scroll bar is enabled. For a selection list, the scroll bar must be enabled to scroll the list. The default value is FALSE.

For browses, the SCROLLBAR-VERTICAL attribute defaults to TRUE unless the DEFINE BROWSE statement Browse Options phrase includes the NO-SCROLLBAR-VERTICAL option. When the vertical scroll bar appears, it appears on the right side of the browse.

For editors and selection lists, you can set this attribute only before the widget is realized.

SEAL( ) method

Seals a Client-principal object with the specified message authentication code (MAC). Sealing a Client-principal object signifies that the user identity from the authentication domain has successfully logged into that domain. Once logged in, you can use the sealed Client-principal object to set a user ID using either the SET-CLIENT( ) method or SET-DB-CLIENT function.

You must set the following attributes on the Client-principal object before you can seal the object:

- USER-ID attribute
- DOMAIN-NAME attribute
- SESSION-ID attribute

If you do not set these attributes, Progress generates a run-time error.

You can seal a Client-principal object only once per user login session. Then, you can use the VALIDATE-SEAL( ) method to validate the seal whenever necessary.

Once sealed, you cannot set any new or existing properties or attributes for the object.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
SEAL( key )
```

key

A character expression containing the key of the authentication domain that authenticated the user ID. Progress uses this key to generate the MAC with which to seal the Client-principal object. Progress converts this key to UTF-8 before using it, which ensures a consistent value regardless of code page settings. A matching authentication domain key must be registered in a trusted authentication domain registry before you can validate and use the user ID represented by the Client-principal object.

If successful, this method returns TRUE. Progress also sets the SEAL-TIMESTAMP attribute with the time stamp of when the Client-principal object was sealed, and sets the LOGIN-STATE attribute to “LOGIN”.

Progress also checks the LOGIN-EXPIRATION-TIMESTAMP attribute. If the Client-principal object expires before you can seal it, Progress sets the LOGIN-STATE attribute to “EXPIRED” and returns FALSE.

Calling this method generates an audit event and creates an audit record for the event in all connected audit-enabled databases according to each database’s current audit policy settings.

Example The following code fragment illustrates how to use the SEAL( ) method:

```
DEF VAR hCP as HANDLE.
DEF VAR key as CHAR.
DEF VAR result as LOGICAL.
.
.
.
CREATE CLIENT-PRINCIPAL hCp.
.
.
.
result = hCP:SEAL(key).
```

See also SEAL-TIMESTAMP attribute, LOGIN-EXPIRATION-TIMESTAMP attribute, LOGIN-STATE attribute, SET-CLIENT( ) method, SET-DB-CLIENT function, VALIDATE-SEAL( ) method

SEAL-TIMESTAMP attribute

Returns the time stamp of when the Client-principal object was sealed, as a DATETIME-TZ. If the Client-principal object is not sealed, this attribute returns the Unknown value (?).

Data type: DATETIME-TZ

Access: Readable

Applies to: Client-principal object handle

See also SEAL( ) method, VALIDATE-SEAL( ) method

SEARCH( ) method

Searches for a specified string in an editor widget starting from the current text cursor position.

Return type: LOGICAL

Applies to: EDITOR widget

Syntax

```
SEARCH ( string , flag )
```

string

The character-string expression for which to search.

Note: You cannot specify wildcard characters in the search string.

flag

An integer expression that specifies the type of search to be performed.

The *flag* expression determines what type of search to perform. Table 84 lists the flag values that correspond to each search type.

Table 84: SEARCH flag values

Type of search	Flag value
FIND-NEXT-OCCURRENCE	1
FIND-PREV-OCCURRENCE	2
FIND-CASE-SENSITIVE	4
FIND-WRAP-AROUND	16
FIND-SELECT	32

For a single search operation, you cannot specify both FIND-NEXT-OCCURRENCE and FIND-PREV-OCCURRENCE. Any other combination of these flags is valid. To do multiple searches, you add the flag values. For example, you can specify FIND-PREV-OCCURRENCE and FIND-WRAP-AROUND by adding their flag values, 2 and 16, to get SEARCH(*string*, 18).

If the operation is successful, the method returns TRUE.

SELECT-ALL( ) method

Selects all rows, or a range of rows, in a query connected to the browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-ALL ( [
  starting-row-table1 , starting-row-table2, ...
  ending-row-table1, ending-row-table2 ...
  ] )
```

starting-row-table1

A variable of type ROWID representing the first row in the first table to select.

starting-row-table2

A variable of type ROWID representing the first row in the second table to select.

ending-row-table1

A variable of type ROWID representing the last row in the first table to select.

ending-row-table2

A variable of type ROWID representing the last row in the second table to select.

If you not specify parameters, the SELECT-ALL method selects all rows.

If you specify the starting row and the ending row, the SELECT-ALL method selects all rows between the starting row and the ending row inclusive.

If the query is a join, a ROWID for each table in the query can be specified for the starting row and the ending row. A maximum of 40 parameters is allowed which allows the user to specify a 20-table join, 20 ROWIDs for the starting row, and 20 ROWIDs for the ending row.

SELECT-FOCUSED-ROW( ) method

Selects the row that currently has focus in a browse widget, even if it is not currently displayed.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-FOCUSED-ROW ( )
```

This method repositions the query to that row and copies the record into the database buffer. The browse automatically scrolls to the selected row. You can use this method after a REPOSITION statement to position a query to a selected row.

SELECT-NEXT-ROW( ) method

Deselects all currently selected rows in a browse and selects the row after the deselected row.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-NEXT-ROW ( )
```

This method also repositions the query to the new row and copies the record into the database buffer. The browse automatically scrolls to the selected row if it is out of view.

This method is intended for use with a browse that supports the selection of a single row at a time (MULTIPLE attribute is set to FALSE). If more than one row is selected when you execute this method, all of the selected rows are deselected and the record after the last selected row becomes the selected row.

SELECT-PREV-ROW( ) method

Deselects a currently selected row in a browse and selects the row before the deselected row.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-PREV-ROW ( )
```

This method also repositions the query to the new row and copies the record into the database buffer. The browse automatically scrolls to the selected row if it is out of view.

This method is intended for use with a browse that supports the selection of a single row at a time (MULTIPLE attribute is set to FALSE). If more than one row is selected when you execute this method, all of the selected rows are deselected and the record before the last selected row becomes the selected row.

SELECT-ROW( ) method

Selects the specified row if it is currently in the browse viewport. In a single-select browse, the previously selected row is deselected. No rows are deselected in a multiple-select browse.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SELECT-ROW ( n )
```

n

An integer expression specifying the ordinal position of a row within the browse.

This method also repositions the query to that row and copies the record into the database buffer.

SELECTABLE attribute (Graphical interfaces only)

Indicates whether a widget is selectable for direct manipulation at run time.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

If the SELECTABLE attribute is TRUE, users can select and deselect the widget (that is, activate SELECTION and DESELECTION events for the widget). You must also set the SELECTABLE attribute to TRUE for the RESIZABLE attribute to take effect, allowing the user to resize the widget.

Note: Setting the SELECTABLE attribute to TRUE enables direct manipulation events for the widget. These events take precedence over all other events. This effectively prevents data entry using the widget until all direct manipulation events are disabled (that is, MOVABLE, RESIZABLE, and SELECTABLE are all set to FALSE). Also, vertical scrollbars are disabled until no direct manipulation can occur (that is, MOVABLE, RESIZABLE, and SELECTABLE are all set to FALSE)

SELECTED attribute (Graphical interfaces only)

Indicates whether a widget is selected (highlighted).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

This attribute can be set in two ways—when the widget's SELECTABLE attribute is TRUE and the user selects the widget, or by setting the SELECTED attribute to TRUE from the 4GL whether or not its SELECTABLE attribute is TRUE. Although setting SELECTED to TRUE from the 4GL highlights the widget, this does not activate a SELECTION event for the widget.

SELECTION-END attribute

The offset of the first character after the end of the currently selected text in the widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

If no text is currently selected, SELECTION-END has the Unknown value (?).

If the editor is not already realized and you reference its SELECTION-END attribute, Progress realizes the widget.

In Windows, both the regular editor and the large editor support SELECTION-END.

SELECTION-START attribute

The offset of the first character of the currently selected text in the widget.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

If the editor is not already realized and you reference its SELECTION-START attribute, Progress realizes the widget.

In Windows, both the regular editor and the large editor support SELECTION-START.

SELECTION-TEXT attribute

The currently selected text in the widget.

Data type: CHARACTER

Access: Readable

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

You can read this attribute to access the text the user has selected. To change or remove the currently selected text, use the REPLACE-SELECTION-TEXT( ) method.

If the editor is not already realized and you reference its SELECTION-TEXT attribute, Progress realizes the widget.

SENSITIVE attribute

Indicates whether a widget can receive input focus or events. Indicates whether certain Progress objects can receive events.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, Server socket object handle, SLIDER widget, Socket object handle, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For widgets, if the SENSITIVE attribute is TRUE, the user can give input focus to the widget or can select, move, or resize it (if other attributes are set). A field-level widget must be specified in a frame before you can set the SENSITIVE attribute. The ENABLE statement implicitly sets this attribute to TRUE, and the DISABLE statement sets it to FALSE.

If the READ-ONLY attribute is TRUE for the widget, the SENSITIVE attribute has no effect except to grey out the widget in some environments.

For the socket and server socket objects, the SENSITIVE attribute indicates whether the object can receive events. The default value of this attribute is TRUE for socket and server socket objects.

If the SENSITIVE attribute is set to FALSE for the socket object, Progress will not execute the READ-RESPONSE procedure for the socket even if the READ-RESPONSE event occurs.

If the SENSITIVE attribute is set to FALSE for the server socket object, Progress will stop accepting connections on the port associated with the server socket.

Note: If an application knows it will not receive data on a socket during some period of time, it should set this attribute to FALSE. This allows the application to run more efficiently since Progress does not monitor the socket if its SENSITIVE attribute is set to FALSE. Data can still be written to an insensitive socket object. When the attribute is set to TRUE, Progress checks the socket for data.

SEPARATORS attribute (Graphical interfaces only)

Indicates whether Progress displays the row and column separators of a browse widget.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget

If SEPARATORS is TRUE, row and column separators appear in the widget. Otherwise, they do not. This attribute can be initialized with the SEPARATORS or NO-SEPARATORS option of the DEFINE BROWSE statement.

SEPARATOR-FGCOLOR attribute (Graphical interfaces only)

Sets the color of a browse's separators.

Data type: INTEGER

Access: Readable/Writable

Applies to: BROWSE widget

If you set a browse's SEPARATOR-FGCOLOR attribute and the separators appear, they have the color you specified.

SERVER attribute

The server handle to the AppServer on which the specified remote persistent procedure runs, or the Web service to which the Web service procedure is bound.

Data type: HANDLE

Access: Readable

Applies to: Asynchronous request object handle, CODEBASE-LOCATOR system handle, THIS-PROCEDURE system handle (and all procedure handles)

For a procedure handle, the SERVER attribute is valid only on a proxy persistent procedure handle that references an active persistent procedure running in the context of an AppServer (that is, where the handle PERSISTENT and PROXY attributes are both set to TRUE). Otherwise, the SERVER attribute is set to the Unknown value (?).

For the CODEBASE-LOCATOR system handle, the SERVER attribute returns the server handle to the connected AppServer to use in accessing application files. Valid only if LOCATOR-TYPE is "AppServer".

To check the validity of a handle, use the VALID-HANDLE function. For more information on the AppServer, see *OpenEdge Application Server: Developing AppServer Applications*.

SERVER attribute

A handle to an AppServer containing logic you want to invoke dynamically.

Data type: HANDLE

Access: Readable/Writable

Applies to: CALL object handle

Syntax

```
SERVER [ = handle-expression ]
```

handle-expression

A HANDLE expression indicating a handle to an AppServer.

The default is the Unknown value (?).

To dynamically invoke logic that resides on an AppServer, set SERVER to the handle of the AppServer.

Note: Do not confuse SERVER and IN-HANDLE. SERVER indicates a handle to an AppServer containing logic you want to invoke dynamically. IN-HANDLE indicates a handle to a running persistent procedure containing logic (internal procedures and user-defined functions) you want to invoke dynamically.

SERVER-CONNECTION-BOUND attribute (AppServer only)

Indicates if the current AppServer agent is bound to a particular client application.

Data type: LOGICAL

Access: Readable

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE.

If the SERVER-CONNECTION-BOUND attribute is TRUE, the current session is bound to a client application. Otherwise, it is FALSE.

On a stateless or state-free AppServer, if the SERVER-CONNECTION-BOUND-REQUEST attribute is set to FALSE to unbind the connection, the connection remains bound and the SERVER-CONNECTION-BOUND attribute remains TRUE as long as remote persistent procedures remain active for the connection.

On a state-aware or state-reset AppServer, the SERVER-CONNECTION-BOUND attribute is always set to TRUE.

SERVER-CONNECTION-BOUND-REQUEST attribute (AppServer only)

Tells an AppServer agent running on a stateless AppServer to bind or unbind its current client connection.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE and the operating mode is stateless.

When set to TRUE, requests that the AppServer agent makes its connection bound to the current client connection identified by the SERVER-CONNECTION-ID attribute.

When set to FALSE, requests that the AppServer agent makes its connection unbound from the currently bound client connection pending the deletion of all remote persistent procedures running in the session. When all remote persistent procedure for the connection have been deleted, the AppServer agent becomes available to service a different client connection. The SERVER-CONNECTION-BOUND attribute for the session is also set to FALSE when the AppServer agent becomes available.

In state-free operating mode, any attempt to set this attribute raises a WARNING condition in the AppServer agent, which writes a message to the AppServer log file, and the value remains unchanged. You can handle the WARNING condition by including the NO-ERROR option in the statement that attempts to set the value, and checking ERROR-STATUS:NUM-MESSAGES for a value greater than zero. In state-free operating mode, this attribute always has the Unknown value (?).

Note: This attribute has no effect on AppServer sessions running in state-aware or state-reset mode.

SERVER-CONNECTION-CONTEXT attribute (AppServer only)

An application-determined value that you set within an AppServer agent. Progress passes this value to each AppServer agent that executes a request on behalf of the client connection identified by the SERVER-CONNECTION-ID attribute.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE.

When a client application requests a connection with an AppServer, the AppServer broker creates an area to store this value for the connection. The initial value is the Unknown value (?).

This attribute, while available in all AppServer operating modes, has practical application only on a stateless AppServer, where more than one AppServer agent can service the same client connection. This value is available to any Connect procedure, Activate procedure, Deactivate procedure, or Disconnect procedure that you have configured for the AppServer, as well as any application procedure. Thus, each AppServer agent that services a client connection can pass context information to the next.

For an AppServer agent, Progress sets the SERVER-CONNECTION-CONTEXT attribute to the Unknown value (?) each time a new connection is assigned to the process. If the AppServer operating mode is state-aware or state-reset, the AppServer agent procedure can also reset this attribute to an application-specific value. However, any such value does not last beyond the current client connection within the current AppServer agent session. Thus, AppServer agents running on a state-aware or state-reset AppServer cannot pass information among themselves using this attribute.

In state-free operating mode, any attempt to set this attribute raises a WARNING condition in the AppServer agent, which writes a message to the AppServer log file, and the value remains unchanged. You can handle the WARNING condition by including the NO-ERROR option in the statement that attempts to set the value, and checking ERROR-STATUS:NUM-MESSAGES for a value greater than zero. In state-free operating mode, this attribute always has the Unknown value (?).

SERVER-CONNECTION-ID attribute (AppServer only)

Returns the run-time connection ID of the current client connection assigned to this AppServer session.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE.

This value is assigned by the AppServer broker when an AppServer accepts a connection request from a client application. The AppServer broker and all AppServer agents use the connection ID as an identifier when they log any information associated with the connection. This same connection ID is available to the AppServer agent using the SERVER-CONNECTION-ID attribute and to the connected 4GL client using the CLIENT-CONNECTION-ID attribute on the server handle connected to this AppServer.

The value of the connection ID is guaranteed to be globally unique for all time within a single computer network. Connection IDs can be compared to each other strictly for equality, but other types of comparisons are irrelevant.

Progress ensures that the SERVER-CONNECTION-ID attribute for each AppServer agent is set to the connection ID for the connection that is assigned to the AppServer agent. Each time a new connection is assigned to an AppServer agent, a new value is assigned to the SERVER-CONNECTION-ID attribute.

This attribute is available to any Connect procedure or Disconnect procedure that you have configured for the AppServer. It maintains the same value for these and all other AppServer procedures executed on behalf of the same connection.

If the AppServer operating mode is stateless, the AppServer broker resets the SERVER-CONNECTION-ID attribute for each AppServer agent to the ID of the connection each time it executes a request. The connection ID for a bound stateless AppServer agent remains the same until the server process becomes unbound and receives a request from a new unbound connection.

If the AppServer operating mode is state-free, this value has no meaning.

SERVER-OPERATING-MODE attribute (AppServer only)

Returns the operating mode of the current AppServer session.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This attribute is valid only if the REMOTE attribute is TRUE.

The possible values for this attribute include:

- `"State-reset"`
- `"State-aware"`
- `"Stateless"`
- `"State-free"`

This is the value of the operatingMode property set for this AppServer in the `ubroker.properties` file. For information on how to configure the operating mode for an AppServer instance, see *OpenEdge Application Server: Administration*.

SESSION-END attribute

Sets the end of a logical WebSpeed session. Intended for internal use only.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

SESSION-ID attribute

Specifies the user login session ID for the user represented by the Client-principal object. You must set this attribute before you can seal the associated Client-principal object using the SEAL( ) method.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

If you specify the Unknown value (?) or the empty string (""), Progress generates a run-time error.

You can also set this attribute to one of the following values:

- The universally unique identifier (UUID) generated by the GENERATE-UUID function.
- The SERVER-CONNECTION-ID attribute, on the SESSION system handle, to map the user ID of the user login session to the connection ID of an AppServer session.

Once the Client-principal object is sealed, this attribute is read-only.

See also SEAL( ) method

SET-ACTOR( ) method

Sets the ACTOR attribute for this SOAP header entry.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-ACTOR ( character )
```

character

A character variable containing the URI of the SOAP actor. The actor can be used to indicate the recipient of a SOAP header element.

Call this method once you have associated the XML with a SOAP header entry using the SET-NODE( ) method.

SET-APPL-CONTEXT( ) method

Sets (and records) the application context for the current session in each connected audit-enabled database whose current audit policy has this audit event enabled. Application context provides meaningful information about the conditions under which an application audit event occurred.

This method returns a Base64 character string that specifies the universally unique identifier (UUID) of the primary index for all audit event records generated by this method for this application context. This UUID is recorded in all subsequent audit event records until you either clear this application context or set a different application context. The UUID is 22 characters in length (the two trailing Base64 pad characters are removed).

Return type: CHARACTER

Applies to: AUDIT-CONTROL system handle

Syntax

```
SET-APPL-CONTEXT( event-context [, event-detail [, user-detail ] ] )
```

event-context

A character expression that specifies the context for the audit event. The value of this expression cannot exceed 200 characters. You can also use this value as an alternate index for querying the audit event record.

If you specify the Unknown value (?), Progress generates a run-time error.

event-detail

An optional character expression that specifies additional audit detail. The value of this expression cannot exceed 10,000 characters.

user-detail

An optional character expression that specifies additional user detail. The value of this expression cannot exceed 10,000 characters.

The UUID is saved as the APPL-CONTEXT-ID attribute value for each connected audit-enabled database.

There can be only one active application context per session at any one point in time. To set a different application context for the session, you can:

- Call the CLEAR-APPL-CONTEXT( ) method, to clear the current application context, and then call the SET-APPL-CONTEXT( ) method with the new application context.
- Call the SET-APPL-CONTEXT( ) method with the new application context. If there is an existing application context in effect, Progress clears the existing application context before setting the new application context.

Calling this method generates an audit event, and creates an audit record for the event in all connected audit-enabled databases according to each database's current audit policy settings.

Example The following code fragment illustrates how to use the SET-APPL-CONTEXT( ) method:

```
DEF VAR name as CHAR.
DEF VAR id as CHAR.
.
.
.
id = AUDIT-CONTROL:SET-APPL-CONTEXT("Payroll app", "fica calculation", name).
.
.
.
AUDIT-CONTROL:LOG-AUDIT-EVENT(34123, "payroll.fica.calc").
.
.
.
AUDIT-CONTROL:CLEAR-APPL-CONTEXT.
```

See also APPL-CONTEXT-ID attribute, CLEAR-APPL-CONTEXT( ) method

SET-ATTRIBUTE( ) method

Adds a new attribute to an element. If an attribute with the same name is already present, its value is replaced with the specified value.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
SET-ATTRIBUTE( name , value )
```

name

A character expression that represents the attribute name.

value

A character expression that represents the attribute value.

The following example creates the following line in the hDoc output .xml file:

```
<Customer Id="54" Name="Second Skin Scuba"/>
```

```
CREATE X-NODEREF hNoderef.
CREATE X-DOCUMENT hDoc.
CREATE X-NODEREF hRoot.

hDoc:CREATE-NODE(hRoot,"root","ELEMENT").
hDoc:APPEND-CHILD(hRoot).
...
hDoc:CREATE-NODE(hNoderef,"Customer","ELEMENT").
hNoderef:SET-ATTRIBUTE("Id","54").
hNoderef:SET-ATTRIBUTE("Name","Second Skin Scuba").
hRoot:APPEND-CHILD(hNoderef).
```

SET-ATTRIBUTE-NODE( ) method

Associates an XML ATTRIBUTE node with the referenced X-noderef object handle.

Return type: LOGICAL

Applies to: X-noderef object handle

Syntax

```
SET-ATTRIBUTE-NODE( attr-node-handle )
```

attr-node-handle

A valid X-noderef handle that represents an XML ATTRIBUTE node created with the CREATE-NODE-NAMESPACE( ) or CREATE-NODE( ) method.

SET-BLUE-VALUE( ) method (Graphical interfaces only)

Specifies the blue component of an entry in the color table.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-BLUE-VALUE ( index , blue-value )
```

index

An integer expression that specifies an entry in the color table.

blue-value

An integer expression that specifies the blue RGB component of an entry in the color table. The value must be in the range 0 to 255.

If the operation is successful, the method returns TRUE.

SET-BREAK( ) method

Sets a breakpoint for a debugging session.

Return type: LOGICAL

Applies to: DEBUGGER system handle

Syntax

```
SET-BREAK ( [ procedure [ , line-number ] ] )
```

procedure

A character expression that specifies the name of the procedure in which you want to set the breakpoint. The specified procedure does not have to exist at the time the breakpoint is set. If you do not specify *procedure*, the method sets the breakpoint at the next executable line of the current procedure.

line-number

An integer expression that specifies the line number in *procedure* (based at line 1 of the debug listing) where you want to remove the breakpoint. A positive integer greater than or equal to 1 represents a line number in the specified *procedure* file. Zero (0) or a negative integer value represents the first executable line of the main procedure block in the specified *procedure* file. If you do not specify *line-number*, the method sets the breakpoint at the first executable line of *procedure* file. If *line-number* is greater than the last executable line number, the method sets the breakpoint at the last executable line of *procedure*. If *line-number* does not specify an executable line, the method sets the breakpoint at the next executable line after the line specified by *line-number*.

If the Debugger is initialized, this method returns TRUE. Otherwise, it returns FALSE with no effect. For more information, see the reference entry for the DEBUGGER system handle.

Note: To use this method, you must have the Application Debugger installed in your OpenEdge environment.

Note that the Debugger sets breakpoints on **physical lines**—not statements. If you invoke DEBUGGER:SET-BREAK( ) on a line that contains other executable statements, all the other statements on that line execute before the breakpoint occurs on the next executable line. This is true whether the statements appear on the same line before or after the invocation of the SET-BREAK( ) method.

If you invoke DEBUGGER:SET-BREAK (*procedure* , *line-number*) on the same line that is specified by *procedure* and *line-number*, the specified line executes the first time without breaking. The breakpoint occurs only on the second and succeeding executions of the line.

Note: You cannot set a watchpoint programmatically using the DEBUGGER system handle. A watchpoint is a form of breakpoint which tells the Debugger to interrupt program execution when the value of a variable, buffer field, or attribute reference changes.

SET-BUFFERS( ) method

Binds all buffers for a query or dynamic ProDataSet object at the same time. Any buffers previously added to the ProDataSet object are removed.

Use the ADD-BUFFER( ) method to add one buffer to the object, without affecting the other buffers, if any.

Return type: LOGICAL

Applies to: ProDataSet object handle, Query object handle

Syntax

```
SET-BUFFERS ( buffer [ , buffer ] ... )
```

buffer

A handle to a buffer, or a CHARACTER expression that evaluates to the name of a buffer that Progress searches for at runtime.

The maximum number of buffers per query is 18.

Note: Searching for a buffer using a handle is more efficient than a character expression. Progress resolves a character expression at runtime by searching in the current routine for a static buffer with that name. If not found, Progress searches the enclosing main procedure. If still not found, Progress searches up through the calling programs of the current routine, and their main procedures. Since a handle uniquely identifies the buffer, no such search is required.

Following is an example:

```
my-query-handle:SET-BUFFERS(BUFFER customer:handle).
```

For more information on buffer objects and query objects, see *OpenEdge Development: Progress 4GL Handbook*.

SET-CALLBACK( ) method

Associates a method within a class object instance, or an internal procedure within a persistent procedure, with a Progress callback event.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Query object handle

Syntax

```
SET-CALLBACK ( callback-name, routine-name [ , routine-context ] )
```

callback-name

A quoted string or character expression representing the name of a callback. The *callback-name* is not case-sensitive.

routine-name

A quoted string or character expression representing the name of a method or an internal procedure that resides within *routine-context*.

routine-context

An object reference for a class object instance or a handle to a persistent procedure that contains the method or internal procedure specified by *routine-name*. If not specified, and the routine is executed within a procedure, THIS-PROCEDURE is used as the routine context. If not specified, and the routine is executed within a class object instance, THIS-OBJECT is used as the routine context.

If *callback-name* is not a valid callback, or *routine-context* is not a valid object reference or handle, this method returns FALSE; otherwise, it returns TRUE. If the SET-CALLBACK( ) method is specified but cannot be invoked, or it fails, no method or procedure is executed as part of the callback event.

SET-CALLBACK-PROCEDURE( ) method

Associates an internal procedure with a Progress callback event.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Query object handle, THIS-PROCEDURE system handle

Syntax

```
SET-CALLBACK-PROCEDURE (callback-name,internal-procedure
                        [ , procedure-context ])
```

callback-name

A quoted string or character expression representing the name of a callback. The *callback-name* is not case-sensitive.

For example:

- For Web services, this can be either "REQUEST-HEADER" or "RESPONSE-HEADER". Progress invokes the internal procedure associated with the “REQUEST HEADER” event as part of an outgoing SOAP request. The request procedure provides access to the SOAP header during the request message. Progress invokes the internal procedure associated with the “RESPONSE HEADER” callback as part of an incoming SOAP request. The response procedure provides access to the SOAP header during the response message.
- For a ProDataSet object, this can be "BEFORE-FILL" or "AFTER-FILL".
- For a ProDataSet object buffer, this can be "BEFORE-FILL", "AFTER-FILL", "BEFORE-ROW-FILL", "AFTER-ROW-FILL", "ROW-CREATE", "ROW-DELETE", "ROW-UPDATE", "FIND-FAILED", or "SYNCHRONIZE".
- For a query, this can be "OFF-END".

You can also use the APPLY-CALLBACK( ) method to apply a callback procedure for an object.

internal-procedure

A quoted string or character expression representing the name of an internal procedure that resides within *procedure-context*.

For Web services, Progress invokes the specified internal procedure as part of the SOAP request message formulation.

procedure-context

A handle to a procedure that contains the internal procedure specified by *internal-procedure*. If not specified, THIS-PROCEDURE is used as the procedure context.

This method returns FALSE if the *callback-name* is not valid or if *procedure-context* is not a valid widget handle; returns TRUE otherwise. If this method is specified but cannot be invoked, or it fails, no procedure will be executed as part of the callback event.

If this method is called multiple times, the new values passed replace the previously set values.

To remove the callback procedure associated with a Web service procedure, invoke this method with a valid *callback-name* and no *internal-procedure* or *procedure-context* parameters specified. Progress will not generate an error if an attempt is made to remove a callback procedure and a callback is not associated with the callback.

For more information on events, see the "Events Reference" section on page 2171.

See also APPLY-CALLBACK( ) method, GET-CALLBACK-PROC-CONTEXT( ) method, GET-CALLBACK-PROC-NAME( ) method

SET-CLIENT( ) method

Uses the user ID associated with a sealed Client-principal object to set the default user ID for the OpenEdge session, and attempts to set the user ID on all connected OpenEdge databases (that do not already have a user ID explicitly set).

Note: If the user ID on one or more connected databases has already been set, by either the SETUSERID or SET-DB-CLIENT functions, the user ID specified by this method is ignored and no attempt is made to reset the user ID on those particular databases. Also, any subsequent calls to either the SETUSERID or SET-DB-CLIENT functions will override the database user ID set by this method.

When a user ID is set on a connected database, Progress uses that user ID to determine whether the user has permission to access tables and fields in that particular database.

Return type: LOGICAL

Applies to: SECURITY-POLICY system handle

Syntax

```
SET-CLIENT( client-principal-handle )
```

client-principal-handle

A handle to a sealed Client-principal object. The Client-principal object must be created in an authentication domain that is registered in the application's trusted authentication domain registry. If the Client-principal object is not sealed, or the handle is the Unknown value (?), Progress generates a run-time error and the current application's user ID remains unchanged.

If the LOGIN-STATE attribute for the sealed Client-principal object is not "LOGIN", Progress generates a run-time error and the current user ID remains unchanged.

If successful, this method returns TRUE. Otherwise, it returns FALSE.

When a user ID is set for an application, and at least one connected audit-enabled database, this method generates an audit event and creates an audit record for the event in all connected audit-enabled databases on which it was set according to each database's current audit policy settings.

You can use this method, instead of the SETUSERID function or the SET-DB-CLIENT function, to set a database user ID when the user ID is not in the _User table. You can also use the SETUSERID function or the SET-DB-CLIENT function to override the user ID set by this method for one or more connected databases.

See also Client-principal object handle, SET-DB-CLIENT function, SETUSERID function

SET-COMMIT( ) method (AppServer only)

Directs the transaction object to commit the transaction when the AppServer session completes the current request and returns execution to the client.

Return type: LOGICAL

Applies to: Transaction object handle

Syntax

```
SET-COMMIT ( )
```

If the operation is successful, the method returns TRUE. If a transaction initiating procedure is not active in the current AppServer session, this method returns FALSE. You also **cannot** invoke this method after prior invocation of a SET-ROLLBACK( ) method during service of the same client request.

SET-CONNECT-PROCEDURE( ) method

Identifies the name of the procedure that is invoked when a CONNECT event occurs.

Return type: LOGICAL

Applies to: Server socket object handle

Syntax

```
SET-CONNECT-PROCEDURE( event-internal-procedure  [ , procedure-context ] )
```

event-internal-procedure

A quoted string or character expression representing the name of an internal procedure that resides within *procedure-context*. When a client has requested a connection to this port, the specified internal procedure is called.

procedure-context

A handle to a procedure that contains the internal procedure specified by *event-internal-procedure*. If not specified, THIS-PROCEDURE is used as the *procedure-context*.

Returns FALSE if *procedure-context* is not a valid widget handle, returns TRUE otherwise. If this method is not invoked, or if it fails, no connection procedure will be executed when the CONNECT event occurs.

For more information on connecting sockets, see *OpenEdge Development: Programming Interfaces*.

SET-DYNAMIC( ) method (Graphical interfaces only)

Sets a color entry to a dynamic or static color.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-DYNAMIC ( index , logical-expr )
```

index

An integer expression that specifies an entry in the color table.

logical-expr

A logical expression that specifies the dynamic status of an entry in the color table.

If *logical-expr* is TRUE and sets the entry to a static color if *logical-exp* is FALSE. If the operation is successful, the method returns TRUE.

SET-GREEN-VALUE( ) method (Graphical interfaces only)

Specifies the green component of an entry in the color table. If the operation is successful, the method returns TRUE.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-GREEN-VALUE ( index , green-value )
```

index

An integer expression that specifies an entry in the color table.

green-value

An integer expression that specifies the green RGB component of an entry in the color table. The value must be in the range 0 to 255.

SET-INPUT-SOURCE( ) method

Specifies the source of the XML to be parsed by the SAX-reader.

Return type: LOGICAL

Applies to: SAX-reader object handle

Syntax

```
SET-INPUT-SOURCE ( mode, { file | memptr | handle | longchar } )
```

mode

A CHARACTER expression evaluating to “FILE”, “MEMPTR”, “HANDLE”, or “LONGCHAR” indicating whether the XML source is a file, a MEMPTR, a WEB-CONTEXT system handle, or a LONGCHAR variable.

file

A CHARACTER expression that indicates the name of a file. This can be a relative pathname, an absolute pathname, or an HTTP URL.

memptr

A MEMPTR variable that contains the loaded XML text. The size of the MEMPTR variable must match the size of the XML text.

handle

A WEB-CONTEXT system handle. In a WebSpeed application, this directs SAX-reader to get the XML source from WEB-CONTEXT.

longchar

A LONGCHAR variable that contains the loaded XML text. The size of the LONGCHAR variable must match the size of the XML text.

The following checks are not performed by SET-INPUT-SOURCE( ) but are performed by SAX-PARSE( ) at run time:

- Whether a file exists and is accessible.
- Whether a MEMPTR is usable.
- Whether WEB-CONTEXT's XML attribute is YES.

  If WEB-CONTEXT is used outside of a WebSpeed environment, Progress raises a run-time error.

SET-MUST-UNDERSTAND( ) method

Sets the MUST-UNDERSTAND attribute for this SOAP header entry.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-MUST-UNDERSTAND ( logical )
```

logical

A logical variable containing the value of the MUST-UNDERSTAND attribute.

Call this method once you have associated the XML with a SOAP header entry using the SET-NODE( ) method.

SET-NODE( ) method

Replaces the header entry referenced by this SOAP header entry with the DOM XML sub-tree rooted by the X-noderef.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-NODE ( x-noderef )
```

x-noderef

A variable of type X-noderef that refers to the XML that will become the SOAP header entry.

Replaces the header entry referenced by this SOAP-header-entryref with the DOM XML sub-tree rooted by X-noderef. The X-noderef will be regarded as corresponding to the header entry element. Performs a deep copy of the X-noderef and its sub-tree and adds namespace declarations to the SOAP-header-entryref as necessary. For example, if the X-noderef uses a namespace that is declared in its parent tree by an ancestor of the X-noderef, that namespace declaration is carried over to the SOAP-header-entryref.

SET-NUMERIC-FORMAT( ) method

Sets the NUMERIC-SEPARATOR and NUMERIC-DECIMAL-POINT attributes simultaneously.

Return type: LOGICAL

Applies to: SESSION system handle

Syntax

```
SET-NUMERIC-FORMAT ( separator , decimal-point )
```

separator

A CHARACTER expression that represents, in formatted text, a number's thousands separator.

The thousands separator cannot be represented by any of the following:

- The characters B C D R Z z 0 1 2 3 4 5 6 7 8 9 + - < > ( ) * ?

  Note: The space character is allowed.

- Any multi-byte character

decimal-point

A CHARACTER expression that represents, in formatted text, a number's decimal point.

The decimal point cannot be represented by any of the following:

- The characters B C D R Z z 0 1 2 3 4 5 6 7 8 9 + - < > ( ) * ?
- The space character
- Any multi-byte character

SET-NUMERIC-FORMAT( ) returns TRUE if the operation is successful.

Note: The values set by the SET-NUMERIC-FORMAT( ) method override the values set by the Thousands Separator (-numsep) and Fractional Separator (-numdec) startup parameters.

SET-OUTPUT-DESTINATION( ) method

Defines the target of the XML document that the SAX-writer object will create.

Return Type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
SET-OUTPUT-DESTINATION ( mode, { file | stream | memptr | longchar } )
```

mode

A character expression that evaluates to one of the following: “FILE”, “STREAM”, “MEMPTR”, or “LONGCHAR”. The *mode* indicates whether the XML target is a file, a stream, a MEMPTR, or a LONGCHAR variable.

file

A character expression that represents the name of a file. You can specify a pathname relative to the current directory or an absolute pathname.

stream

A character expression that represents the name of a 4GL stream. If stream is "", Progress saves the document to the unnamed stream of the 4GL session.

memptr

A MEMPTR variable that the XML document will be written to. The MEMPTR will be overwritten and the new size of the MEMPTR variable will match the size of the XML text.

longchar

A LONGCHAR variable that the XML document will be written to. The LONGCHAR will be overwritten and the new size of the LONGCHAR variable will match the size of the XML text.

When writing an XML document to a LONGCHAR variable, Progress writes the LONGCHAR variable in the code page of the XML document as determined by the XML document's ENCODING attribute. If the ENCODING attribute is not set, the LONGCHAR variable is saved in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE function), the code page must be equivalent to the encoding specified in the XML document's ENCODING attribute. If not, the START-DOCUMENT( ) method returns an error and the XML document is not saved to the LONGCHAR variable.

Use this method to set the output destination, which is where the object will write the XML document. This method must be called before you call any of the writing methods or they will raise errors. You can only call this method when the object's WRITE-STATUS property is SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. In other words, you cannot set a new output destination while the SAX-writer object is currently writing XML. This method fails and generates an error message if it is called while writing.

The SET-OUTPUT-DESTINATION method does not check if the specified destination is valid. If does not check whether a MEMPTR or LONGCHAR variable is usable, and it does not check whether a file location or stream is accessible. This destination is checked at run time by the START-DOCUMENT method.

When writing to a MEMPTR or LONGCHAR, the method deletes the previous contents and allocates new memory. For example, writing to a MEMPTR is the logical equivalent of using SET-SIZE based upon the document size after calling END-DOCUMENT. This is the same way that the X-document object handles memory.

SET-PARAMETER( ) method

Lets you set parameters of one of the following:

- A procedure or user-defined function you want to invoke dynamically.
- An attribute you want to get or set dynamically.
- A method you want to invoke dynamically.

Return type: LOGICAL

Applies to: CALL object handle

Syntax

```
SET-PARAMETER(parameter-number, data-type, iomode, parameter-value)
```

parameter-number

An INTEGER expression indicating the order of the parameter. Use 1 for the first parameter, 2 for the second parameter, and so on.

data-type

A CHARACTER expression indicating the data type of the parameter and evaluating to one of the following:

- "CHARACTER"
- "DATASET-HANDLE"
- "DATE"
- "DATETIME"
- "DATETIME-TZ"
- "DECIMAL"
- "HANDLE"

- “INTEGER”
- “LOGICAL”
- “LONGCHAR”
- “MEMPTR”
- “RAW”
- “ROWID”
- “TABLE-HANDLE”

TABLE-HANDLE can be a handle to a static or dynamic temp-table. Likewise, DATASET-HANDLE can be a handle to a static or dynamic ProDataSet object.

Note: To manage BUFFER parameters, use HANDLE parameters.

For each parameter, the data type specified by the caller and the callee must be compatible.

iomode

A CHARACTER expression indicating the mode of the parameter and evaluating to one of the following:

- “INPUT”
- “OUTPUT”
- “INPUT-OUTPUT”
- “OUTPUT-APPEND”

For each parameter, the mode specified by the caller and the callee must match.

For parameters of attributes, specify “INPUT.”

For TABLE-HANDLE output parameters whose output is to be appended to the table, and in no other case, specify “OUTPUT-APPEND.”

If *data-type* is “DATASET-HANDLE”, you can append "`-by-reference`" to any *iomode* listed above.

parameter-value

An expression whose type is compatible with *data-type*.

The *parameter-value* argument may represent a determinate or indeterminate array. However, you cannot pass an array by value. You can pass an array only as a variable defined using the DEFINE VARIABLE statement with the EXTENT option. If you include the EXTENT keyword in the *data-type* argument, Progress ignores it. If you include the array element values in the *parameter-value* argument, Progress generates a compile error.

If *iomode* is "INPUT" or "INPUT-OUTPUT," if the data type of *parameter-value* does not agree with the data type passed, *parameter-value* will automatically be converted to the data type passed during SET-PARAMETER( ) processing.

If iomode is "OUTPUT" or "INPUT-OUTPUT," each of the following must be true:

- *parameter-value* must represent a program variable or a NO-UNDO TEMP-TABLE field (perhaps with an array reference).
- *parameter-value* must still be in scope and must still be valid when the dynamic invoke is executed.

  If the output value from the called procedure does not have a data type that matches the data type passed, the output value will be converted to the data type passed.

If iomode is "OUTPUT," *parameter-value* is ignored if the ASYNCHRONOUS attribute is TRUE.

SET-PROPERTY( ) method

Sets the value of the specified application-defined property associated with an unsealed Client-principal object.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
SET-PROPERTY( property-name, property-value )
```

property-name

A character string that specifies the name of an application-defined property associated with the Client-principal object. You must enclose this character string in quotes. If you specify the Unknown value (?) or the empty string (""), Progress generates a run-time error.

property-value

A character expression that contains the value for the specified property. You must enclose this character expression in quotes. If you specify the Unknown value (?) or the empty string (""), Progress sets the property to that value (and you cannot change it).

If successful, this method returns TRUE. Otherwise, it returns FALSE.

If you call this method more than once per property, Progress generates a run-time error.

Once all property values are set for the Client-principal object, you must seal the object using the SEAL( ) method. Once sealed, you cannot set any new or existing properties for the object. If you call this method for a sealed Client-principal object, Progress generates a run-time error.

Calling this method does not generate an audit event or an audit record.

You can use the GET-PROPERTY( ) method to get the value of a single property associated with a Client-principal object, or use the LIST-PROPERTY-NAMES( ) method to retrieve a list of all properties associated with the Client-principal object.

Example The following code fragment illustrates how to use the SET-PROPERTY( ) method:

```
DEF VAR hCP as HANDLE.
DEF VAR val-ok as LOGICAL.
.
.
.
CREATE CLIENT-PRINCIPAL hCP.
hCP:SET-PROPERTY("eye-color", "Blue").
.
.
.
val-ok = hCP:VALIDATE-SEAL(key).
```

See also GET-PROPERTY( ) method, LIST-PROPERTY-NAMES( ) method

SET-READ-RESPONSE-PROCEDURE( ) method

Specifies the name of the procedure to invoke when a READ-RESPONSE event occurs.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
SET-READ-RESPONSE-PROCEDURE( event-internal-procedure
                             [ , procedure-context ] )
```

event-internal-procedure

A quoted string or character expression representing the name of an internal procedure that resides within *procedure-context*. When data is available on the socket, the specified internal procedure is called. If not specified, then no read procedure will be executed when the READ-RESPONSE event occurs.

procedure-context

A handle to a procedure that contains the internal procedure specified by *event-internal-procedure*. If not specified, THIS-PROCEDURE is used as the *procedure-context*.

Note: Returns FALSE if the *procedure-context* is not a valid widget handle; returns TRUE otherwise. If this method is not invoked, or it fails, no read procedure will be executed when the READ event occurs.

SET-RED-VALUE( ) method (Graphical interfaces only)

Specifies the red component of an entry in the color table.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-RED-VALUE ( index , red-value )
```

index

An integer expression that specifies an entry in the color table.

red-value

An integer expression that specifies the red RGB component of an entry in the color table. The value must be in the range 0 to 255.

If the operation is successful, the method returns TRUE.

SET-REPOSITIONED-ROW( ) method

Sets the row index where records positioned with the REPOSITION TO ROWID (or RECID) statement are displayed.

Return type: LOGICAL

Applies to: BROWSE widget

Syntax

```
SET-REPOSITIONED-ROW ( n , "ALWAYS" | "CONDITIONAL" )
```

n

Indicates the row number where the new record is displayed, 1 being the first row.

"ALWAYS"

Specifies that the REPOSITION TO ROWID statement always uses the indicated row number.

"CONDITIONAL"

Specifies that the REPOSITION TO ROWID statement uses the indicated row number unless the new row is already in the browse viewport. In this case, the REPOSITION statement moves focus to the existing row.

By default, this is the top row in the browse viewport (index 1). If the associated query is defined with the INDEXED-REPOSITION option, the CONDITIONAL option is ignored.

SET-RGB-VALUE( ) method (Graphical interfaces only)

Specifies a combination of the red, green, and blue values of an entry in the color table.

Return type: LOGICAL

Applies to: COLOR-TABLE system handle

Syntax

```
SET-RGB-VALUE ( index , int-value )
```

index

An integer expression that specifies an entry in the color table.

int-value

An integer expression that specifies the RGB component of an entry in the color table. You can obtain this value from the color property of an ActiveX control, by using the RGB-VALUE function, or by using the GET-RGB-VALUE( ) method.

If the operation is successful, the method returns TRUE.

SET-ROLLBACK( ) method (AppServer only)

Directs the transaction object to rollback the transaction when the AppServer session completes the current request and returns execution to the client.

Return type: LOGICAL

Applies to: Transaction object handle

Syntax

```
SET-ROLLBACK ( )
```

If the operation is successful, the method returns TRUE. If a transaction initiating procedure is not active in the current AppServer session, this method returns FALSE. You also **can** invoke this method after prior invocation of a SET-COMMIT( ) method during service of the same client request.

SET-SELECTION( ) method

Selects (and highlights) the text in a widget between two specified character offsets.

Return type: LOGICAL

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

Syntax

```
SET-SELECTION ( start-pos , end-pos )
```

start-pos

An integer expression that specifies the offset of the first character to be selected.

end-pos

An integer expression that specifies the offset of the first character after the selection.

This method selects the text that begins at the offset *start-pos* and ends at the offset *end-pos* . If the operation is successful, the method returns TRUE. Otherwise, it returns FALSE.

Note: This operation produces a different result depending on the platform. In Windows GUI platforms, Progress measures character offset positions **between** characters. On non-Windows GUI or character platforms, Progress measures character offset positions **on** characters.

In Windows, both the regular editor and the large editor support SET-SELECTION.

SET-SERIALIZED( ) method

Sets the SOAP header entry's underlying XML from serialized XML.

Return type: LOGICAL

Applies to: SOAP-header-entryref object handle

Syntax

```
SET-SERIALIZED ( longchar )
```

longchar

A variable of type LONGCHAR that contains the serialized XML that will become the SOAP-header-entryref object.

Functions the same as SET-NODE( ), except it expects a LONGCHAR whose contents is equivalent to the serialized form of the X-noderef that would be passed to the SET-NODE( ) method. No validation is done by this method. If the contents of the LONGCHAR is not valid for the SOAP message, an error might be raised when the SOAP message is processed. If the LONGCHAR is empty, or the caller passes the Unknown value (?), the header entry will be removed from the SOAP message.

SET-SOCKET-OPTION( ) method

Sets the specified socket option. TCP supports a number of socket options. Please refer to TCP documentation for a description of these options.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
SET-SOCKET-OPTION( name , arguments )
```

name

A character expression which indicates the name of the socket option to be set.

arguments

A character expression that contains a comma separated list of arguments specific for the option.

Table 85 describes the options Progress supports.

Table 85: Options for the SET-SOCKET-OPTION( ) method *(1 of 2)*

Option	Description
TCP-NODELAY	An *enable indicator*, which is either TRUE or FALSE.
SO-LINGER	Two comma separated values: • The *onoff indicator*, which is either TRUE or FALSE. • The *linger time*. If the onoff indicator is FALSE, the linger time does not need to be provided.
SO-KEEPALIVE	Sets the TCP socket option SO_KEEPALIVE. Set *arguments* to TRUE to turn this option on or to FALSE to turn it off.
SO-REUSEADDR	Sets the TCP socket option SO_REUSEADDR. Set *arguments* to TRUE to turn this option on or to FALSE to turn it off.

Table 85: Options for the SET-SOCKET-OPTION( ) method *(2 of 2)*

Option	Description
SO-RCVBUF SO-SNDBUF	Sets the TCP socket option SO_RCVBUF or SO_SNDBUF. Set *arguments* to the desired size of the buffer. **Note:** Depending on your platform, the value you supply might be increased to the platform's minimum buffer size, decreased to the platform's maximum buffer size, or rounded up to the next multiple of the platform's segment size. For more information, see your platform's documentation.
SO-RCVTIMEO	Sets the timeout length—that is, the number of seconds the socket waits to receive data before timing out. Set *arguments* to the desired timeout value in seconds. If a timeout occurs, READ( ) returns TRUE and the value of BYTES-READ is zero. This is true whether the READ( ) mode is READ-AVAILABLE or READ-EXACT-NUM. For more information on the interaction of READ( ), the READ( ) mode, and SO_RCVTIMEO, see *OpenEdge Development: Programming Interfaces*.

The SET-SOCKET-OPTION( ) method returns TRUE if setting the option succeeded and returns FALSE otherwise. An error can occur if:

- *name* is not a Progress supported socket option.
- The arguments supplied for the option are not valid.
- The SET-SOCKET-OPTION( ) operation fails.

SET-WAIT-STATE( ) method

Sets or cancels a Progress wait state that blocks user and system input.

Return type: LOGICAL

Applies to: SESSION system handle

Syntax

```
SET-WAIT-STATE ( state-string )
```

state-string

A character-string expression that sets the wait state.

The value of *state-string* determines the state and the type of wait message displayed. The valid values are:

- "GENERAL", which displays a general working message.
- "COMPILER", which displays a message that Progress is compiling.

Input is blocked and the message is displayed until the wait state is cancelled. The null string ("") cancels the wait state.

The SET-WAIT-STATE( ) method accepts an arbitrary mouse pointer name (any string which is a valid argument to the LOAD-MOUSE-POINTER( ) method) as an argument, in addition to the “GENERAL”, “COMPILER”, and “” states. The return value is TRUE if the wait-state is set successfully; otherwise the return value is FALSE.

Note that this method is intended to provide user feedback for lengthy processing that involves no user input, such as compiling procedures, doing a time consuming database lookup, or some long CPU and memory operation like computing the value of π.

This method is not supported in character mode.

If an error occurs from this method, Progress displays the error and terminates the wait state.

Caution: Be sure that the processing you invoke after setting the wait state is guaranteed to cancel the wait state. Otherwise, Progress remains in the wait state indefinitely. For example, do not place user input statements, such as SET or UPDATE, between the setting and cancelling of the wait state. Because the user cannot respond to these statements during the wait state, Progress I/O blocks indefinitely, preventing the wait state from being cancelled.

SHOW-IN-TASKBAR attribute (Windows only)

Determines whether an icon for the window appears in the taskbar and in the task-switching window displayed when **ALT+TAB** is pressed.

Data type: LOGICAL

Access: Readable/Writable

Applies to: WINDOW widget

Applications that display several windows might want only the main window to have an icon on the taskbar.

This attribute defaults to TRUE. If SMALL-TITLE is set to TRUE, the SHOW-IN-TASKBAR attribute will be set to FALSE because, in general, a tool palette should not have an icon in the taskbar. You can override this behavior by setting the SHOW-IN-TASKBAR attribute to TRUE after setting the SMALL-TITLE attribute to TRUE.

If a window that does not appear in the taskbar is minimized, Windows shrinks the window so only the title bar is visible. Windows displays the window at the bottom of the screen. This is standard behavior, but might be unexpected to people who are used to finding minimized windows in the taskbar.

The SHOW-IN-TASKBAR attribute must be set before the window is realized.

SIDE-LABEL-HANDLE attribute

A handle to the side label of a widget.

Data type: WIDGET-HANDLE

Access: Readable/Writeable

Applies to: COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TEXT widget

For static widgets, this attribute is read-only and the handle accesses a literal widget containing the side label specified when the widget was defined. For dynamic widgets, you can set this handle to a text widget that you create as a side label. You first must create a dynamic text widget to use as a label (assign it a value, row, and column); then assign the handle of the text widget to the SIDE-LABEL-HANDLE attribute of the widget whose label you want to specify or change.

SIDE-LABELS attribute

Indicates whether a frame displays labels to the left of each field.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget

The SIDE-LABELS attribute returns TRUE if the frame displays labels to the left of each field rather than above each field.

SKIP-DELETED-RECORD attribute

Indicates whether Progress should skip deleted records when accessing a dynamic query's result list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Query object handle

SMALL-ICON attribute

Returns the name of the icon loaded by LOAD-SMALL-ICON( ).

Data type: CHARACTER

Access: Readable

Applies to: WINDOW widget

SMALL-TITLE attribute

Indicates whether the window has a palette-style title bar.

Data type: LOGICAL

Access: Readable/Writable

Applies to: WINDOW widget

This title bar is shorter than a normal Windows title bar, and is commonly used for tool palettes (such as in the AppBuilder) and other auxiliary windows. Windows with small title bars do not have maximize or minimize buttons; they only have close buttons.

The MIN-BUTTON and MAX-BUTTON attributes have no effect on a window with a small title bar and are ignored. The CONTROL-BOX attribute specifies whether the window has a close button and system menu (available by right-clicking on the title bar or by pressing **ALT+SPACE**).

The SMALL-TITLE attribute must be set before the window is realized. The default value of SMALL-TITLE is FALSE.

SOAP-FAULT-ACTOR attribute

The URI of the Web service actor that caused this SOAP fault.

Data type: CHARACTER

Access: Readable

Applies to: SOAP-fault object handle

SOAP-FAULT-CODE attribute

Identifies the SOAP fault code for this SOAP-fault object.

Data type: CHARACTER

Access: Readable

Applies to: SOAP-fault object handle

SOAP-FAULT-DETAIL attribute

Returns the handle of the SOAP fault detail information associated with this SOAP-fault object.

Data type: HANDLE

Access: Readable

Applies to: SOAP-fault object handle

SOAP-FAULT-STRING attribute

Returns the SOAP fault string describing the fault for this SOAP-fault object.

Data type: CHARACTER

Access: Readable

Applies to: SOAP-fault object handle

SORT attribute

Indicates whether to sort new additions to the item list of a widget.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: COMBO-BOX widget, SELECTION-LIST widget

If the SORT attribute is TRUE, all items added to a combo box or selection list are added in sorted order. This means that the methods ADD-FIRST( ) and ADD-LAST( ) add items to the list in sorted order. The setting of this attribute has no affect on the function of the INSERT( ) and REPLACE( ) methods. Setting this attribute to FALSE returns these methods to their native function.

SSL-SERVER-NAME attribute

The name of the server for the current Secure Sockets Layer (SSL) session.

Data type: CHARACTER

Access: Readable

Applies to: Server object handle, Socket object handle

For the AppServer or a socket object, this is the digital certificate subject name of the server for the current SSL session. It enables you to distinguish between the physical host name and the authenticated SSL server name.

For Web services, this is the digital certificate subject name of the `-SOAPEndpoint` (which is the URL identifying the endpoint for the Web service, not the server providing the WSDL).

When there is no socket connection, or the socket connection is not an SSL-based connection, the default value is the Unknown value (`?`).

STANDALONE attribute

Determines the value of the standalone string in the XML declaration of a SAX-writer object.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SAX-writer object handle

You can set the attribute to the value of the standalone string in the XML declaration. The default value is the Unknown value (?). If the value is the Unknown value (?) then the standalone string will not appear in the XML declaration.

Valid values of `standalone` in the XML declaration are “yes” and “no”. If you set the `standalone` value, then the standalone string appears in the XML declaration.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

This attribute can be read at all times, but can only be written when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, it can only be changed when the writer is not writing, otherwise it will fail and generate an error message.

START-DOCUMENT( ) method

Creates the XML document with the prolog information.

Note: You must call the SET-OUTPUT-DESTINATION( ) method to set the output destination for the XML document before invoking this method.

Return Type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
START-DOCUMENT()
```

You must call this method to start the output before you call any other writing methods or none of the writing methods will succeed. After you call START-DOCUMENT, the WRITE-STATUS is changed to SAX-WRITE-BEGIN. If START-DOCUMENT is called while the SAX-writer is already writing (that is, with a status other than SAX-WRITE-IDLE or SAX-WRITE-COMPLETE), then the method fails.

If the SAX-writer is unable to write to the output destination, it generates an error message and change the WRITE-STATUS to SAX-WRITE-ERROR.

If the FRAGMENT attribute is FALSE, then the XML declaration is created. The version, encoding style, and standalone state of the document is specified in the declaration. If the ENCODING attribute is not set, the declaration defaults to UTF-8, but this value does not appear in the declaration. If the VERSION attribute is not set, it defaults to “1.0”. If the STANDALONE attribute is not set, then the declaration default is to omit the attribute. Here is the default declaration:

```
<?xml version="1.0"?>
```

See also END-DOCUMENT( ) method, SET-OUTPUT-DESTINATION( ) method

START-ELEMENT( ) method

Starts an XML node based upon the name of the node in a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
START-ELEMENT( name [, namespace-URI ] )
```

name

A LONGCHAR expression evaluating to the fully qualified or unqualified name of the element.

namespace-URI

A LONGCHAR expression evaluating to the URI of the element, or an empty string ("") or the Unknown value (?) if the element doesn't contain a namespace.

Starts an XML node. This method call changes the WRITE-STATUS to SAX-WRITE-TAG.

For every call of the START-ELEMENT method, there must be a following corresponding call of the END-ELEMENT method. All the parameter values from the two calls must match for the methods to correspond.

If *namespace-URI* is present, then the prefix will be resolved in the following order:

1. The method attempts to extract the namespace from the *name*.
2. The method attempts to extract the namespace from a previously declared namespace.
3. The method attempts to generate the default namespace.

If the name contains a prefix, *namespace-URI* is present, and this is the first instance of the *namespace-URI*, then the namespace is added to the element. This technique is equivalent to calling the DECLARE-NAMESPACE method.

If only the *name* is present and it contains a prefix, then the SAX-writer attempts to resolve the prefix to a namespace.

If the STRICT attribute is TRUE, the FRAGMENT attribute is FALSE, and the call would result in more than one document-level element (that is, root node), then the method fails. Also, if STRICT is TRUE, an external DTD has been declared, and the call would create the root node, then the name used for the DTD declaration must match the name of the root node or the method fails.

See also END-ELEMENT( ) method

STARTUP-PARAMETERS attribute

Returns a character string containing a comma-separated list of all startup parameters you defined at startup for the current OpenEdge session.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

This list includes startup parameters defined in the Progress default startup parameter file (`$DLC/startup.pf`) or the file specified by the `$PROSTARTUP` environment variable, as well as startup parameters you specify on the command line or within a parameter file (`.pf`). The value of this attribute does not change during runtime.

The startup parameter values in this list reflect initial parameter value settings. The value of a startup parameter during an OpenEdge session can be different from its initial value setting. Some startup parameters have an equivalent session attribute you can use in 4GL code to override the parameter value during a session. If you override a parameter value using an equivalent session attribute, the new value is not reflected in the list of startup parameters returned in this attribute.

If you defined any startup parameters in the default parameter file (`startup.pf`), or another parameter file specified by the Parameter File (`-pf`) startup parameter, the list includes the `-pf` *`filename`* parameter and all parameters defined in that parameter file, followed by `(end .pf)`. For example:

```
-pf dbconnect.pf,-db sports2000,-H pclsmith,-S 5000,(end .pf)
```

A parameter file appears in the list whether or not it contains startup parameters. If a parameter file does not contain startup parameters, it appears in the list in the following format:

```
-pf filename,(end .pf)
```

The default parameter file (`startup.pf`) always appears in the list. Progress expands the filename of only the default parameter file. All other filenames appear in the list as specified.

Individual startup parameters, defined within a parameter file or on the command line, appear in the list in the following format:

```
-parameter-name parameter-value,
```

If the startup parameter has no value, the list contains the startup parameter followed by a comma. No space appears before or after a comma, and no comma appears at the end of the list.

If the list of startup parameters includes duplicates, the last occurrence takes precedence and all other instances are ignored (even though they appear in the list).

If the list of startup parameters includes the Password (`-P`) or Proxy Password (`-proxyPassword`) parameters, Progress substitutes six asterisks in place of the password value.

If the list of startup parameters contains a hyphen with no parameter name, the hyphen is ignored.

You can use the ENTRY function to parse the list of startup parameters. If you use the ENTRY function with the default delimiter (comma), the function separates the parameter entries wherever a comma appears. If a comma appears in the list as part of a parameter value, the function might not parse the list correctly. A comma separating two startup parameters, as opposed to being part of a parameter value, is always followed by "`-`" or "`(end .pf)`". Based on this convention, you can examine the character(s) after a comma to determine whether the comma is separating two startup parameters or is part of a parameter value.

If you started your OpenEdge session with the Statistics (`-y`), Statistics with **CTRL+C** (`-yc`), or Segment Statistics (`-yd`) startup parameter, you can use the SHOW-STATS statement to see the value of the STARTUP-PARAMETERS attribute. This statement includes the value of this attribute in the output to the `client.mon` file.

Table 86 shows examples of original command lines and their equivalent STARTUP-PARAMETERS attribute values.

Table 86: STARTUP-PARAMETERS attribute usage examples *(1 of 2)*

Original command	Value of the STARTUP-PARAMETERS attribute
`prowin32 -db sports2000 -T c:\temp -H pclsmith -S 5000` where `startup.pf` contains no startup parameters	`-pf c:\dlc\startup.pf,(end .pf),-db sports2000,-T c:\temp,-H pclsmith,-S 5000`

`prowin32 -T c:\temp`
 	
where `startup.pf` contains:
 	
`-db sports2000`
 	
`-H pclsmith`
 	
`-S 5000`
	`-pf c:\dlc\startup.pf,-db sports2000,-H pclsmith,-S 5000,(end .pf),-T c:\temp`

`prowin32 -pf dbconnect.pf -T c:\temp`
 	
where `startup.pf` contains no startup parameters and `dbconnect.pf` contains:
 	
`-db sports2000`
 	
`-H pclsmith`
 	
`-S 5000`
	`-pf c:\dlc\startup.pf,(end .pf),-pf dbconnect.pf,-db sports2000,-H pclsmith,-S 5000,(end .pf),-T c:\temp`

Table 86: STARTUP-PARAMETERS attribute usage examples *(2 of 2)*

Original command	Value of the STARTUP-PARAMETERS attribute

`prowin32 - -T d:\work100a -db mystore -1 -db corporate -H corpmachine -S 5000`
 	
where `startup.pf` contains:
 	
`-T c:\temp`
	`-pf c:\dlc\startup.pf,-T c:\temp,(end .pf),-T d:\work100a,-db mystore,-1,-db corporate,-H corpmachine,-S 5000`

`prowin32 -U lsmith -P mypassword`
 	
where `PROSTARTUP=c:\commonarea\dbconnect.pf` and `c:\commonarea\dbconnect.pf` contains:
 	
`-db sports2000`
 	
`-H pclsmith`
 	
`-S 5000`
	`-pf c:\commonarea\db.pf,-db sports2000,-H pclsmith,-S 5000,(end .pf),-U lsmith,-P ******`

STATE-DETAIL attribute

A description that provides detail about the current state of the Client-principal object.

Data type: CHARACTER

Access: Readable

Applies to: Client-principal object handle

Progress sets the value of this attribute, along with the LOGIN-STATE attribute, whenever the state of a Client-principal object changes. You can also set the value of this attribute when calling the AUTHENTICATION-FAILED( ) method to place a Client-principal object in an authentication failed state by specifying a reason for the authentication failure.

See also AUTHENTICATION-FAILED( ) method, LOGIN-STATE attribute

STATUS-AREA attribute

Indicates whether a window has a status area.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WINDOW widget

If the STATUS-AREA attribute is TRUE, the window has a status area.

You can set this attribute only before the window is realized.

STATUS-AREA-FONT attribute (Graphical interfaces only)

The font number of the font used in the status area of a window

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

The font number represents an entry in the font table maintained by the FONT-TABLE handle.

STOP attribute

Set to TRUE, if the asynchronous request was executing when the client issued the CANCEL REQUESTS( ) method.

Data type: LOGICAL

Access: Readable

Applies to: Asynchronous request object handle

If the COMPLETE attribute is FALSE, the value of this attribute is the Unknown value (?). When the PROCEDURE-COMPLETE event is processed, this attribute is set to TRUE before the event procedure is executed if the remote request returned with an unhandled STOP condition; otherwise, it is set to FALSE.

STOP-PARSING( ) method

Causes the parser to stop parsing the XML document. This lets an application search for particular data, then abort the parse as soon as the data are found.

Return type: LOGICAL

Applies to: SAX-reader object handle

Syntax

```
STOP-PARSING ( )
```

STOP-PARSING( ) can stop a parse started by SAX-PARSE( ), SAX-PARSE-FIRST( ) or SAX-PARSE-NEXT( ). That is, the parse can be single call or multiple scan.

STOP-PARSING sets the PARSE-STATUS attribute to SAX-COMPLETE.

Within a callback, to invoke STOP-PARSING( ), use the SELF system handle, as shown in the following fragment:

```
SELF:STOP-PARSING( ).
```

If STOP-PARSING( ) is invoked in a callback or in any procedure called directly or indirectly by a callback, Progress continues to execute the callback as usual, but when the callback finishes, control returns to the next 4GL statement after the most-recently-executed SAX-PARSE( ), SAX-PARSE-FIRST( ), or SAX-PARSE-NEXT( ).

STOPPED attribute

Indicates whether the last compilation stopped prior to completion.

Data type: LOGICAL

Access: Readable

Applies to: COMPILER system handle

When set to TRUE, the STOPPED attribute indicates that the last Progress 4GL compilation stopped before completion.

STREAM attribute

A value that specifies the character set used for operating system file I/O — "`ibm850`" or "`iso8859-1`".

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

The Stream Character Set (`-stream`) parameter sets the value of this attribute.

This attribute is obsolete. See the CPSTREAM attribute.

STRETCH-TO-FIT attribute

Forces the image to expand or contract to fit within the image widget's boundaries.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: IMAGE widget

This attribute has no effect if an icon is displayed on the image widget.

STRICT attribute

Determines if the SAX-writer object should ensure that the XML document is well formed XML.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SAX-writer object handle

The default value, TRUE, directs the object to ensure that the output is well formed XML. If a method call results in invalid XML, then the method fails, WRITE-STATUS is changed to SAX-WRITE-ERROR, and the stream is closed.

FALSE directs the object to generate warning messages and proceed with the write. If any warning message is generated, then the XML document will not be valid.

You can read this attribute at all times, but you can only write to it when the object's WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, it can only be changed when the writer is not writing, otherwise it fails and generates an error message.

STRING-VALUE attribute

The string value (which Progress computes at run time) of the contents of the buffer-field object.

The STRING-VALUE attribute uses the format attribute to convert the buffer value to a string.

Data type: CHARACTER

Access: Readable

Applies to: Buffer-field object handle

Syntax

```
STRING-VALUE [ ( i ) ]
```

i

An INTEGER expression representing a subscript, for fields that have extents.

SUBTYPE attribute

The subtype of a widget.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: COMBO-BOX widget, FILL-IN widget, MENU-ITEM widget, Server object handle, X-document object handle, X-noderef object handle

This attribute is read-only for combo boxes, the server object handle, the X-document object handle, and the X-noderef object handle. You can set this attribute only before the widget is realized.

For menu items, the value of this attribute is either "NORMAL", "SKIP", or "RULE". "NORMAL" is the default—a menu item that can be chosen, a toggle-box item, or read-only text. (In this case the READ-ONLY and TOGGLE-BOX attributes determine the specific type of the menu item.) "SKIP" specifies a blank line in the menu. "RULE" specifies a visible horizontal line in the menu.

For combo boxes, the value of this attribute is either "SIMPLE", "DROP-DOWN", or "DROP-DOWN-LIST". The DROP-DOWN-LIST subtype is the default. The SIMPLE and DROP-DOWN subtypes apply only to character-field or character-variable combo-box widgets in graphical interfaces only, and only in Windows. If you set the subtype of a combo-box widget to “SIMPLE” or “DROP-DOWN” in a character interface, Progress treats the combo-box widget as having the “DROP-DOWN-LIST” subtype.

For fill-ins, the value of this attribute is either "PROGRESS" or "NATIVE". "PROGRESS" is the default. If set to "PROGRESS", the fill-in widget has the behavior of a standard Progress field in character mode. Otherwise, the field has the behavior of a fill-in that is native to the current graphical environment. The NATIVE option of the VIEW-AS phrase specifies that the field adhere to the native behavior of the current window system or environment.

For the X-document object handle or X-noderef object handle, this attribute returns the name of the object type (character representation of the DOM NodeType), which will be one of the following: ATTRIBUTE, CDATA-SECTION, COMMENT, DOCUMENT, DOCUMENT-FRAGMENT, ELEMENT, ENTITY-REFERENCE, PROCESSING-INSTRUCTION, or TEXT.

For the server object handle, this attribute identifies the type of server to which the server object is bound. This is either an AppServer or a Web service. This attribute is set during the execution of the CONNECT( ) method and can be one of three values. Before you invoke the CONNECT( ) method, the attribute value is set to the empty string “”. Once you invoke the CONNECT( ) method, the attribute value is set to either “APPSERVER” for an AppServer or “WEBSERVICE” for a Web service.

SUPER( ) method

Invokes the constructor method for a super class.

Return type: VOID

Applies to: All user-defined classes.

Syntax

```
SUPER ( [ parameter [, parameter ] ... ] ).
```

```
( parameter [, parameter ] ... )
```

Specifies one or more parameters of the constructor method.

For the parameter passing syntax, see the Parameter passing syntax reference entry in this book.

Notes

If the constructor method for the super class takes parameters, the first executable statement in the constructor method for a subclass must invoke the constructor method for the super class using this method. The constructor method for the subclass must also provide the parameters identified by the constructor method for the super class with respect to the number, data type, and mode.

If the constructor method for the super class does not take parameters, you need not invoke it using this method. Progress automatically invokes the constructor method when it instantiates the class object.

See also CONSTRUCTOR statement

SUPER-PROCEDURES attribute

A list of the super procedure handles associated with a procedure file or with the current OpenEdge session. The handles appear in last in first out (LIFO) order, comma-delimited, in character format. Returns the empty string for a Web service procedure.

For more information on super procedures and procedure overriding, see *OpenEdge Development: Progress 4GL Handbook*.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle, THIS-PROCEDURE system handle (and all procedure handles)

If there are no super procedures associated with a procedure file or with the current OpenEdge session, the value of the SUPER-PROCEDURES attribute is the empty string.

SUPPRESS-NAMESPACE-PROCESSING attribute

Indicates whether namespace processing is suppressed.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SAX-reader object handle, X-document object handle

FALSE, the default, indicates that namespace processing is not suppressed. TRUE indicates that namespace processing is suppressed.

For more information on accessing XML documents using the Document Object Model (DOM) and Simple API for XML (SAX) interfaces, see *OpenEdge Development: Programming Interfaces*.

SUPPRESS-WARNINGS attribute

Indicates whether Progress suppresses warning messages during the session.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, Progress does not display warning messages during the session.

SYMMETRIC-ENCRYPTION-ALGORITHM attribute

A character string that specifies the name of the default cryptographic algorithm to use with the ENCRYPT and DECRYPT functions. The default value is “AES_CBC_128”.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

This string is a concatenation of three character expressions that identify an algorithm, mode, and key size. For a list the supported cryptographic algorithms, see the SYMMETRIC-SUPPORT attribute reference entry.

You are responsible for generating, storing, and transporting this value.

SYMMETRIC-ENCRYPTION-IV attribute

The default initialization vector value to use with the encryption key in the ENCRYPT and DECRYPT functions. The default value is the Unknown value (?), which indicates that no initialization vector value is used .

Data type: RAW

Access: Readable/Writeable

Applies to: SECURITY-POLICY system handle

Using an initialization vector value increases the strength of the specified encryption key (that is, it makes the key more unpredictable).

You are responsible for generating, storing, and transporting this value.

SYMMETRIC-ENCRYPTION-KEY attribute

The default encryption key (a binary value) to use with the ENCRYPT and DECRYPT functions. The default value is the Unknown value (?).

Data type: RAW

Access: Writeable

Applies to: SECURITY-POLICY system handle

You may specify this key as a MEMPTR, CHARACTER, or LONGCHAR value, but Progress treats it as a RAW.

If the value of this attribute is the Unknown value (?), you must provide the encryption key as an argument to the ENCRYPT and DECRYPT functions.

Progress compares the size of the specified encryption key to the key size specified by the cryptographic algorithm. If the key sizes are inconsistent, Progress generates a run-time error.

Progress obscures this attribute value to protect it against unauthorized access. You are responsible for generating, storing, and transporting this value.

You can generate an encryption key, based on the PKCS#5/RFC 2898 standard, by using either the GENERATE-PBE-KEY function or the GENERATE-RANDOM-KEY function.

Note: Do not use the GENERATE-RANDOM-KEY function to assign a key value to this attribute directly. Doing so will render the key irretrievable (as this attribute is write-only).

SYMMETRIC-SUPPORT attribute

Returns a comma-separated list of supported cryptographic algorithm names to use in encrypting and decrypting data. Each algorithm name is a concatenation of three character expressions that identify an algorithm, mode, and key size.

Data type: CHARACTER

Access: Readable

Applies to: SECURITY-POLICY system handle

Table 87 lists the supported cryptographic algorithm names.

Table 87: Supported cryptographic algorithm names

AES_CBC_128	AES_OFB_192
AES_CBC_192	AES_OFB_256
AES_CBC_256	DES_CBC_56
AES_CFB_128	DES_CFB_56
AES_CFB_192	DES_ECB_56
AES_CFB_256	DES_OFB_56
AES_ECB_128	DES3_CBC_168
AES_ECB_192	DES3_CFB_168
AES_ECB_256	DES3_ECB_168
AES_OFB_128	DES3_OFB_168

SYNCHRONIZE( ) method

Synchronizes a hierarchy of data-relation queries on a parent buffer.

Return type: LOGICAL

Applies to: Buffer object handle

Syntax

```
hBuff:SYNCHRONIZE ( )
```

This method traverses the ProDataSet object hierarchy starting at buffer *hBuff* and reopens each data-relation query for the current parent at each lower level. Use this method to populate one or more related child buffers for the ProDataSet object buffer.

By default, if the query is associated with a browse, the synchronize action automatically refreshes the browse. If the query is not associated with a browse, the synchronize action automatically gets the first buffer in the query by invoking a GET FIRST operation. If there is a REPOSITION data relation and no browse, the synchronize action gets the next record in the query by invoking a GET NEXT operation.

SYSTEM-ALERT-BOXES attribute

Indicates whether Progress displays system messages in alert boxes.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, Progress displays system messages in alert boxes rather than in the message area.

SYSTEM-ID attribute

Returns the system ID of the external DTD from which an XML document was generated. This contains the path to the DTD which is either a file system path or an HTTP URL. The Progress parser uses this information to retrieve the DTD when parsing the document.

Data type: CHARACTER

Access: Readable

Applies to: X-document object handle

TAB-POSITION attribute

The tab order of a widget within its field group.

Data type: INTEGER

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

You can change the tab order of the widget at the field level using the MOVE-BEFORE-TAB-ITEM( ) or MOVE-AFTER-TAB-ITEM( ) methods, and at the field group level using the FIRST-TAB-ITEM attribute or LAST-TAB-ITEM attribute.

TAB-STOP attribute

Returns TRUE if the widget is in its parent's tab chain.

Data type: LOGICAL

Access: Readable/Writable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

Setting the TAB-STOP attribute to FALSE removes the widget from its parent's tab chain. Setting the TAB-STOP attribute to TRUE adds the widget to the end of its parent's tab chain. If the widget is already in the tab chain, its position does not change.

In Windows, the mnemonic key (**ALT** accelerator) for a widget will not work if the widget is removed from the tab order. Also, because the widget is not in the tab order, pressing **TAB** will not change focus from the widget.

TABLE attribute

The name of the database table containing the field associated with a widget, buffer, or buffer-field.

Data type: CHARACTER

Access: Readable

Applies to: BROWSE widget (column), Buffer object handle, Buffer-field object handle, COMBO-BOX widget, EDITOR widget, FILL-IN widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Note: The TABLE attribute of a buffer contains the name of the table, not the name of the buffer.

TABLE-CRC-LIST attribute

Returns a comma-separated list of the CRC value for each table referenced in the r-code file specified by the RCODE-INFO:FILE-NAME attribute.

Data type: CHARACTER

Access: Readable

Applies to: RCODE-INFO system handle

This list corresponds directly to the list generated by the TABLE-LIST attribute.

Use this attribute with the TABLE-LIST attribute to compare the CRC value for all tables referenced in the r-code file with those stored in the database to determine whether or not a procedure file needs to be recompiled after a database change.

If the r-code file was compiled without table references, this attribute returns the empty string (““).

TABLE-HANDLE attribute

Returns the handle of a temp-table object, if any, associated with the buffer object. If the buffer is not associated with a temp-table object, it returns the Unknown value (?).

Data type: WIDGET-HANDLE

Access: Readable

Applies to: Buffer object handle

This attribute allows you to delete a default buffer object for a temp-table object by deleting the temp-table object (since it is illegal to delete the default buffer object itself.)

TABLE-LIST attribute

Returns a comma-separated list of all tables referenced in the r-code file specified by the RCODE-INFO:FILE-NAME attribute. Each table reference includes the table name and related database name (for example: `SPORTS.CUSTOMER`).

Data type: CHARACTER

Access: Readable

Applies to: RCODE-INFO system handle

This list corresponds directly to the list generated by the TABLE-CRC-LIST attribute.

Use this attribute with the TABLE-CRC-LIST attribute to compare the CRC value for all tables referenced in the r-code file with those stored in the database to determine whether or not a procedure file needs to be recompiled after a database change.

If the r-code file was compiled without table references, this attribute returns the empty string (“ ”).

TABLE-NUMBER attribute

The sequence number, within the database, of the table that corresponds to a buffer.

Data type: INTEGER

Access: Readable

Applies to: Buffer object handle

Tag property (Windows only; Graphical interfaces only)

A variable that lets the developer store an arbitrary string value.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Any ActiveX control

The Tag property is an extended ActiveX control property that lets the developer store an arbitrary string value and retrieve it later. Progress does not use this property internally; rather, the property lets the developer store application-specific information with the control.

This property is initialized to an empty string.

Note: The length of the string cannot exceed 2,147,483, 647 characters.

TEMP-DIRECTORY attribute

The name of the directory in which Progress stores temporary files during the session.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

By default, this is the current working directory. Otherwise, it is the directory specified using the Temporary Directory (-T) parameter.

TEMP-TABLE-PREPARE( ) method

Signifies that all the field and index definitions for a temp-table have been supplied.

Return type: LOGICAL

Applies to: Temp-table object handle

Syntax

```
TEMP-TABLE-PREPARE( temp-table-name-exp )
```

temp-table-name-exp

A character expression that evaluates to a temp-table name to be used in subsequent query statements that refer to this temp-table.

The temp-table is in an UNPREPARED state after the first definitional method is called until this method is called. During this time, only ADD/CREATE type methods may be called.

The TEMP-TABLE-PREPARE( ) method must be called after all fields and indexes have been created and before any non-ADD/CREATE method can be called. This method causes the pending list of field and index definitions to become part of the actual temp-table object, which puts the temp-table in a PREPARED state (that is, makes it ready for use).

TEXT-SELECTED attribute

Indicates whether text is currently selected in a widget.

Data type: LOGICAL

Access: Readable

Applies to: BROWSE widget (column), COMBO-BOX widget, EDITOR widget, FILL-IN widget

The TEXT-SELECTED attribute is TRUE if text in the widget is currently selected.

THREE-D attribute (Windows only; Graphical interfaces only)

Indicates whether Progress displays widgets using a three-dimensional format.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, SESSION system handle, WINDOW widget

If TRUE, the specified widgets are displayed in a three-dimensional format. For a frame or dialog box, any field-level widgets in the frame or dialog box are also displayed in three-dimensional format. If a frame has the THREE-D attribute set to TRUE, the default background color is the color Button Face rather than the color Window. For a window, setting this attribute changes the window background color to color Button Face only, and has no effect on any widgets contained in the window. Frames do not inherit the THREE-D attribute from a window or ancestor frame.

If the THREE-D attribute is TRUE for the SESSION handle, then all system dialog boxes and alert boxes are displayed in three-dimensional format.

You can set this attribute only before the widget is realized.

Note: To maintain size compatibility, Progress sets the default vertical size of two-dimensional fill-ins equal to the vertical size of three-dimensional fill-ins. Also, Progress does not fully support the overlay of three-dimensional widgets. For more information, see the section on three-dimensional layout in *OpenEdge Development: Programming Interfaces*.

TIC-MARKS attribute (Windows only; Graphical interfaces only)

Enables the display of short hash marks on the outside of a slider to help indicate the movement of the trackbar with the slider widget. The default is not to display tic marks. If you specify the TIC-MARKS option, it is assumed that you are using new code to create a slider, and the trackbar on the slider widget will be relatively large.

However, if you omit the TIC-MARKS option, the 4GL assumes that you are migrating old code, and the default size of the slider is the size originally defined for the slider in the old code.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SLIDER widget

If you want to use the large trackbar but do not want tic marks to display, specify TIC-MARKS NONE.

To implement the TIC-MARKS option, you must also specify on which side, or sides, of the trackbar tic-marks appear by using the additional qualifying values. Table 88 lists and defines these values.

Table 88: TIC-MARK values

Value	Description
TOP	TIC-MARKS appear on the top of the slider only.
BOTTOM	TIC-MARKS appear on the bottom of the slider only.
LEFT	TIC-MARKS appear on the left side of the slider only.
RIGHT	TIC-MARKS appear on the right side of the slider only.
BOTH	TIC-MARKS appear on both sides of the slider.

The TIC-MARKS attribute must be set before the slider is realized. Also, you can use the FREQUENCY attribute with the TIC-MARKS attribute to indicate how frequently a tic mark will display along the trackbar of a slider.

TIME-SOURCE attribute

Specifies the client or database server machine that serves as the time source for applications running during the OpenEdge session.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SESSION system handle

TIME-SOURCE accepts either of the following settings:

- "`local`" or the null string ("")

  Your application uses the client machine as its time source. The default value is "".

- "*dbname*"

  Your application uses the machine running the server for the database with the name *dbname* as its time source.

All time-related language elements, such as the MTIME, NOW, TIME, TIMEZONE, and TODAY functions, use the specified time source. This attribute is useful for client/server applications that span time zones.

TITLE attribute

The title string a widget displays.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only), WINDOW widget

For browse widgets, pop-up menus, and frames, this attribute is writeable only before the widget is realized. However, you can modify an existing frame title after realization.

TITLE-BGCOLOR attribute (Graphical interfaces only)

The color number for the background color of the widget title.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle. This attribute is read-only for browse widgets and dialog boxes. In Windows, this attribute is read-only for all applicable widget types.

TITLE-DCOLOR attribute (Character interfaces only)

The color number for the character-mode display color of the widget title

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle. This attribute is read-only for browse widgets.

TITLE-FGCOLOR attribute (Graphical interfaces only)

The color number for the foreground color of the widget title.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only)

The color number represents an entry in the color table maintained by the COLOR-TABLE handle. This attribute is read-only for browse widgets and dialog boxes. In Windows, this attribute is read-only for all applicable widget types.

TITLE-FONT attribute

The font number for the font of the widget title.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, DIALOG-BOX widget, FRAME widget, MENU widget (pop-up only)

The font number represents an entry in the font table maintained by the FONT-TABLE handle. This attribute is read-only for browse widgets and dialog boxes. In Windows, this attribute is also read-only for frames. For menus, this attribute is writeable only before realization.

TOGGLE-BOX attribute

Indicates whether a menu-item appears and acts like a toggle box.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: MENU-ITEM widget

If the TOGGLE-BOX attribute is TRUE, the menu item appears and interacts like a toggle box. You can set this attribute only before the widget is realized.

TOOLTIP attribute (Windows only; Graphical interfaces only)

A help text message for a text field or text variable. Progress automatically displays this text when the user pauses the mouse pointer over a widget for which a tooltip is defined.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, EDITOR widget, FILL-IN widget, IMAGE widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, and TOGGLE-BOX widget

You can add or change the TOOLTIP attribute at any time. If TOOLTIP is set to "" or the Unknown value (?), then the ToolTip is removed. No ToolTip is the default.

TOOLTIPS attribute (Windows only; Graphical interfaces only)

Indicates whether ToolTip information is displayed when the mouse pointer pauses over a control for which tooltip information is defined.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If TRUE, the ToolTip information that is defined for any controls associated with a given session displays when the mouse pointer pauses over a control. Otherwise, ToolTip information does not display for any controls in the session.

Top property (Windows only; Graphical interfaces only)

The vertical position of the control-frame and control-frame COM object from the top border of the parent container widget, in pixels.

Return type: INTEGER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the ROW attribute and Y attribute of the corresponding control-frame widget to an equivalent value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.

TOP-ONLY attribute

Indicates whether another frame or window can overlay a frame or window.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: FRAME widget, WINDOW widget

If the TOP-ONLY attribute is TRUE for the frame, no other frame can overlay it.

If more than one window is designated as TOP-ONLY, they will all stay on top of all non-TOP-ONLY windows, but each can be brought to the foreground. That is, a TOP-ONLY window is always on top of all non-TOP-ONLY windows, but is not necessarily on top of all TOP-ONLY windows.

The TOP-ONLY behavior will be temporarily suspended while a dialog box is displayed to prevent the TOP-ONLY windows from covering the dialog-box.

A window cannot have both the TOP-ONLY and ALWAYS-ON-TOP attributes set to TRUE. Setting the TOP-ONLY attribute to TRUE will set the ALWAYS-ON-TOP attribute to FALSE. The default value of the TOP-ONLY attribute is FALSE.

TRACKING-CHANGES attribute

Set to TRUE to start tracking changes to the data in an individual ProDataSet temp-table. Progress tracks changes to the temp-table until you set this attribute to FALSE. When this attribute is FALSE, any changes you make to the data in the temp-table are considered part of the fill process. The default value is FALSE.

For all other temp-tables, the value of this attribute is the Unknown value (?).

Data type: LOGICAL

Access: Readable/Writeable

Applies to: Temp-table object handle

When the TRACKING-CHANGES attribute is set to TRUE for a ProDataSet temp-table, Progress tracks changes to the data in that temp-table using a before-image table that contains the original version of each row. You can think of the temp-table itself as the after-image because it contains the latest version of each row.

Every row in the after-image table that has been modified or created corresponds to a row in the before-image table. Deleted rows do not appear in the after-image table, because it reflects the current state of the data. Every row in the before-image table has a non-zero ROW-STATE attribute value, because every row in the before-image table is the before-image of a deleted, created, or modified row in the after-image table. Unchanged rows do not appear in the before-image table.

You can track newly created rows and changed rows through either the before-image table or the after-image of the table. However, since deleted rows do not appear in the after-image table, it is better to track changes through the before-image table.

Set TRACKING-CHANGES back to FALSE for a temp-table when you are ready to:

- Accept the changes using the ACCEPT-CHANGES( ) method or the ACCEPT-ROW-CHANGES( ) method.
- Reject the changes using the REJECT-CHANGES( ) method or the REJECT-ROW-CHANGES( ) method.
- Get and merge the changes using the GET-CHANGES( ) method and the MERGE-CHANGES( ) method or MERGE-ROW-CHANGES( ) method, respectively.

TRANSACTION attribute

A handle to the current transaction object. Returns the Unknown value (?) for a Web service procedure.

Data type: HANDLE

Access: Readable

Applies to: THIS-PROCEDURE system handle (and all procedure handles)

The transaction handle returned by this attribute provides attributes and methods that allow you to manage a transaction object running on an AppServer. In an OpenEdge client session, or in an AppServer session that has no active transaction initiating procedure, you can only use the IS-OPEN attribute to check whether a transaction is open.

For more information on the AppServer and transaction initiating procedures, see the TRANSACTION-MODE AUTOMATIC statement reference entry and *OpenEdge Application Server: Developing AppServer Applications*. For more information on the attributes and methods provided by the transaction handle, see the Transaction object handle reference entry in the "Handle Reference" section on page 1379.

TRANSPARENT attribute

Makes the background color of the image transparent. The background color is determined by the color of the pixel in the lower-left corner of the image.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: IMAGE widget

The TRANSPARENT attribute overrides the CONVERT-3D-COLORS attribute; if both TRANSPARENT and CONVERT-3D-COLORS are set to TRUE, CONVERT-3D-COLORS is ignored.

This attribute has no effect if an icon is displayed on the image widget.

TRANS-INIT-PROCEDURE attribute (AppServer only)

The handle to the transaction initiating procedure that started the currently-open automatic transaction.

Data type: HANDLE

Access: Readable

Applies to: Transaction object handle

You can use this procedure handle to access the attributes and methods of the active transaction initiating procedure or to delete the procedure, thus terminating the automatic transaction.

If no automatic transaction is active, TRANS-INIT-PROCEDURE returns an invalid handle. To check a handle for validity, use the VALID-HANDLE function.

For information on automatic transaction initiating procedures, see the TRANSACTION-MODE AUTOMATIC statement reference entry. For more information on the AppServer, see *OpenEdge Application Server: Developing AppServer Applications*.

TYPE attribute

The type of a handle.

Data type: CHARACTER

Access: Readable

Applies to: Asynchronous request object handle, BROWSE widget (browse and cell), Buffer object handle, Buffer-field object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, ProDataSet object handle, Data-relation object handle, Data-source object handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, LOG-MANAGER system handle, MENU widget, MENU-ITEM widget, Query object handle, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget, CLIPBOARD system handle, CODEBASE-LOCATOR system handle, COLOR-TABLE system handle, COMPILER system handle, CURRENT-WINDOW system handle, DEBUGGER system handle, DEFAULT-WINDOW system handle, ERROR-STATUS system handle, FILE-INFO system handle, FOCUS system handle, FONT-TABLE system handle, LAST-EVENT system handle, RCODE-INFO system handle, SAX-attributes object handle, SAX-reader object handle, SELF system handle, Server object handle, Server socket object handle, SESSION system handle, SOAP-fault object handle, SOAP-fault-detail object handle, SOAP-header object handle, SOAP-header-entryref object handle, Socket object handle, THIS-PROCEDURE system handle (and all procedure handles), WEB-CONTEXT system handle, X-document object handle, X-noderef object handle

The TYPE attribute returns the widget or handle type. Some examples are "WINDOW", "FRAME", "BUTTON," “MENU,” “SAX-ATTRIBUTES,” “SAX-READER,” and “SERVER.”

For AppServer and Web service handles, the TYPE attribute returns "SERVER".

If a system handle (such as CURRENT-WINDOW or FOCUS) refers to a user interface widget, the TYPE attribute returns the type of that widget. If a system handle (such as SESSION or CLIPBOARD) refers to a Progress status or system widget, the TYPE attribute value is "PSEUDO-WIDGET".

For procedure handles and system handles that refer to procedures (such as THIS-PROCEDURE), the TYPE attribute returns "PROCEDURE".

For an asynchronous request handle, the TYPE attribute returns "ASYNC-REQUEST".

For the ProDataSet, Data-relation, and Data-source object handles, this attribute returns "DATASET", "DATA-RELATION", and "DATA-SOURCE", respectively.

For a server-socket handle, the TYPE attribute returns "SERVER-SOCKET", and for a socket handle, it returns "SOCKET".

For the SOAP-fault and SOAP-fault-detail object handles, this attribute returns "SOAP-FAULT" and "SOAP-FAULT-DETAIL", respectively.

For the SOAP-header and SOAP-header-entryref object handles, this attribute returns "SOAP-HEADER" and "SOAP-HEADER-ENTRYREF", respectively.

For the X-document and X-noderef object handles, the TYPE attribute returns "X-DOCUMENT" and "X-NODEREF", respectively.

UNDO attribute

If TRUE, the temp-table is UNDO; if FALSE, the temp-table is NO-UNDO. The default is FALSE (NO-UNDO). This attribute can only be updated before the TEMP-TABLE-PREPARE( ) method has been called.

Data type: LOGICAL

Access: Readable/Writable

Applies to: Temp-table object handle

UNIQUE-ID attribute

A value that Progress guarantees is unique within the OpenEdge session.

Data type: INTEGER

Access: Readable

Applies to: Buffer object handle, Buffer-field object handle, ProDataSet object handle, Query object handle, SAX-attributes object handle, SAX-reader object handle, SOAP-header object handle, SOAP-header-entryref object handle, THIS-PROCEDURE system handle (and all procedure handles), X-document object handle, X-noderef object handle

Progress reserves the right to recycle procedure handles within an OpenEdge session. If your application runs persistent procedures, stores the handles, deletes the procedures, and runs more persistent procedures, either the same ones or different ones, the procedure handles you stored might now correspond to different persistent procedure instances.

To avoid this problem, store the UNIQUE-ID attribute of the procedure handle along with the handle. Before you reuse a stored procedure handle, compare the value of UNIQUE-ID that you stored against the value of the UNIQUE-ID attribute of the stored handle. If they do not match, you know that Progress recycled the procedure handle, and you can discard it before it causes damage.

UNIQUE-MATCH attribute (Windows only; Graphical interfaces only)

Specifies that the combo-box widget automatically complete keyboard input based on a unique match to items in the drop-down list.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: COMBO-BOX widget

When the UNIQUE-MATCH attribute is TRUE, the widget's edit control compares the input to the items in the drop-down list. After each incremental character keystroke, the edit control searches through the items in the drop-down list for a unique match. When a unique match is found, the full item is displayed in the edit control. The automatically completed portion of the item is highlighted. You can replace the highlighted portion of the item by typing over it, or you can delete the highlighted portion of the item using the **DELETE** key or the **BACKSPACE** key. The default value is FALSE.

URL attribute

A URL to connect to an AppServer, through the AppServer Internet Adapter (AIA), or a web server.

Data type: CHARACTER

Access: Readable

Applies to: CODEBASE-LOCATOR system handle

Valid URL protocols depend on the LOCATOR-TYPE. If LOCATOR-TYPE is "AppServer", valid URL protocols include: HTTP, HTTPS, and AppServer. If LOCATOR-TYPE is "InternetServer", valid URL protocols include: HTTP, HTTPS, and FILE.

URL-DECODE( ) method

Returns a URL string to decode. This method is called by the `url-decode` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

URL-ENCODE( ) method

Returns characters to encode. This method is called by the `url-encode` WebSpeed API function. Intended for internal use only.

Return type: CHARACTER

Applies to: WEB-CONTEXT system handle

URL-PASSWORD attribute

Password parameter for connecting to the server referenced in the URL, if required by the URL protocol.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

URL-USERID attribute

Userid parameter for connecting to the server referenced in the URL, if required by the URL protocol.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CODEBASE-LOCATOR system handle

USER-ID attribute

The user ID associated with a Client-principal object. You must set this attribute before you can seal the associated Client-principal object using the SEAL( ) method.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Client-principal object handle

If you specify the Unknown value (?) or the empty string (""), Progress generates a run-time error.

Once the Client-principal object is sealed, this attribute is read-only.

V6DISPLAY attribute (Windows only)

Indicates whether Progress follows Version 6 rules or Version 7 rules when it lays out and displays widgets in Windows. This attribute lets you compile and execute Progress Version 6 applications on Progress Version 7 in Windows.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: SESSION system handle

If the V6DISPLAY attribute is TRUE, Progress uses Version 6 rules to manage the display:

- The default font is the default system fixed pitch font (overridable using the DefaultFixedFont parameter in the current environment, which might be the Registry (Windows only) or an initialization file).
- All fill-ins have no borders.

- Fill-ins enabled for input use an underline version of the system fixed pitch font (overridable using the DefaultUpdateFont parameter in the current environment, which might be the Registry (Windows only) or an initialization file).
- The default window size (row/column) is 25 by 80 (overridable in the current environment, which might be the Registry (Windows only) or an initialization file).

Note: PUT SCREEN output is not restorable in graphical environments.

To run an application with V6DISPLAY set to TRUE, you must compile the application with the V6DISPLAY set to TRUE.

Note: The OpenEdge ADE toolset was not compiled or designed to run in V6DISPLAY mode. Running the OpenEdge ADE in V6DISPLAY mode may result in clipped display elements and other unexpected behavior.

Setting V6DISPLAY to TRUE when running the OpenEdge ADE toolset may also degrade application compilation performance.

This attribute provides the same functionality as the V6Display parameter in the current environment, which might be the Registry (Windows only) or an initialization file. For more information on environments, see the chapter on user interface environments in *OpenEdge Deployment: Managing 4GL Applications*.

VALIDATE( ) method

Executes any validation tests established in a database or specified by the VALIDATE option of the Format phrase.

Return type: LOGICAL

Applies to: BROWSE widget (browse and cell), COMBO-BOX widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, RADIO-SET widget, SELECTION-LIST widget, SLIDER widget, TOGGLE-BOX widget

Syntax

```
VALIDATE ( [ "ENABLED-FIELDS" ] )
```

"ENABLED-FIELDS"

Validate enabled fields only.

If this option does not appear, the VALIDATE method validates all fields, whether enabled or not.

For a supported field-level widget, this method executes the validation test associated with the underlying field or variable.

For a frame or dialog box, this method executes the validation tests for every supported field-level widget in the frame or dialog box (except the browse, which you must VALIDATE explicitly). If the test for any field-level widget in the frame fails, Progress displays the validation message and gives focus to the first widget in the frame or dialog box that is both visible and sensitive and whose data has failed validation.

For a browse, VALIDATE executes all validation tests associated with the browse and its children.

Note: During data entry, any widget that receives input focus is always validated. This method allows your procedure to validate any and all widgets in a frame, whether or not they currently have input focus.

If the validation is successful, the method returns TRUE. Otherwise, it returns FALSE.

VALIDATE-EXPRESSION attribute

The value of the validation expression in the database schema for the database field that corresponds to the buffer-field.

The VALIDATE-EXPRESSION attribute lets you write user input validation code for interfaces that Progress's automatic user input validation does not support.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

If a buffer-field is associated with a dynamic browse column, you should set the buffer-field's VALIDATE-EXPRESSION attribute before the dynamic browse column is added to the browser (via ADD-LIKE-COLUMN( )). The validation expression is compiled at this time. If the VALIDATE-EXPRESSION attribute is changed later, it is ignored.

VALIDATE-MESSAGE attribute

The value of the validation message in the database schema for the database field that corresponds to the buffer-field.

The VALIDATE-MESSAGE attribute lets you write user input validation code for interfaces that Progress's automatic user input validation does not support.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

VALIDATE-SEAL( ) method

Validates the message authentication code (MAC) generated by the SEAL( ) method to seal a Client-principal object.

You can use this method to validate the seal whenever necessary.

Return type: LOGICAL

Applies to: Client-principal object handle

Syntax

```
VALIDATE-SEAL( [ validation-key ] )
```

validation-key

An optional character expression containing the authentication domain's key to use in validating the MAC that sealed the Client-principal object. Progress converts this key to UTF-8 before using it, which ensures a consistent value regardless of code page settings.

If you specify a validation key, Progress uses that key to validate the seal. If you do not specify a validation key, Progress uses the authentication domain's key stored in the application's trusted authentication domain registry for the Client-principal object to validate the seal. Progress validates the seal by comparing it to the MAC generated by either the specified validation key or the authentication domain key stored in the trusted authentication domain registry. If the seal matches the MAC, then the seal is valid and this method returns TRUE. Otherwise, the seal is invalid and this method returns FALSE.

If the Client-principal object is not sealed and not in the LOGIN state, Progress generates a run-time error.

Progress also checks the LOGIN-EXPIRATION-TIMESTAMP attribute. If the Client-principal object expires before you can validate its seal, Progress sets the LOGIN-STATE attribute to "EXPIRED" and returns FALSE.

Calling this method does not generate an audit event or an audit record.

Example The following code fragment illustrates how to use the VALIDATE-SEAL( ) method:

```
DEF VAR hCP as HANDLE.
DEF VAR key as CHAR.
DEF VAR val-ok as LOGICAL.
.
.
.
CREATE CLIENT-PRINCIPAL hCp.
.
.
.
val-ok = hCP:VALIDATE-SEAL(key).
```

See also LOGIN-EXPIRATION-TIMESTAMP attribute, LOGIN-STATE attribute, SEAL( ) method

VALIDATE-XML attribute

Sets validation on parsing when an XML document is posted to the transaction server. The default is NO.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

VALIDATION-ENABLED attribute

Indicates whether the parser validates the XML document against the DTD.

Data type: LOGICAL

Access: Readable/Writable

Applies to: SAX-reader object handle

TRUE indicates that the parser validates the XML document against the DTD. The default is TRUE.

Note: If VALIDATION-ENABLED is FALSE, the parser still checks that the XML document is well formed.

VALUE attribute

The data values in the system clipboard.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: CLIPBOARD system handle

During single-item operation (with the CLIPBOARD:MULTIPLE attribute set to FALSE), reading this attribute returns all the data in the clipboard, and writing to attribute completely replaces any and all data in the clipboard. During multiple-item operation (with the CLIPBOARD:MULTIPLE attribute set to TRUE), reading this attribute returns one of several data items in the clipboard, and writing to this attribute appends a data item to a buffered value formatted to replace the data in the clipboard.

Note: In Windows, the clipboard can store a maximum of 64K of data.

If there is no data in the clipboard or the last data item has been read during a multiple-item operation, the VALUE attribute returns the Unknown value (?). Reading this attribute during either single-item or multiple-item operation has no effect on the existing clipboard data value(s). Setting the VALUE attribute to the Unknown value (?) during single- or multiple-item operation has no effect. To clear the clipboard of all data, set the VALUE attribute to the null string in a single-item operation. For more information on using the VALUE attribute, see the reference entry for the CLIPBOARD system handle.

VERSION attribute

Determines the value of the version string in the XML declaration of a SAX-writer object.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: SAX-writer object handle

You can set the attribute to the value of the version string in the XML declaration. The default value is "1.0". For example:

```
<?xml version="1.0" encoding="UTF-8"?>
```

You can read this attribute at all times, but you can only write to it when the WRITE-STATUS is either SAX-WRITE-IDLE or SAX-WRITE-COMPLETE. That is, it can only be changed when the writer is not writing, otherwise it fails and generates an error message.

If STRICT is TRUE, the only valid version is "1.0", and the VERSION attribute cannot be changed or you get an error message. If STRICT is FALSE, the version in the prolog of the XML document will match the value of the VERSION attribute.

If the value is an empty string (""), then the version string will not appear in the XML declaration.

VIEW-FIRST-COLUMN-ON-REOPEN attribute

Controls whether the browse, when an OPEN Query statement is run, displays the first column in the viewport or the columns that were in the viewport before the Query was reopened.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget

When the VIEW-FIRST-COLUMN-ON-REOPEN attribute is set to:

- TRUE, and the query for a browse is reopened, the browse displays the first row of data and the first column in the leftmost position.
- FALSE, and the query for a browse is reopened, the browse displays the first row of data and the columns that were displayed in the viewport before the query was reopened.

  For example, if this attribute is set to FALSE and a user had scrolled to the far-right column, the next time the browse for a query is reopened, the browse displays the first row of data and the far-right column.

The default value is FALSE.

VIRTUAL-HEIGHT-CHARS attribute

The maximum height of the widget, in character units.

Data type: DECIMAL

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a non-scrollable frame, VIRTUAL-HEIGHT-CHARS has the same value as the HEIGHT-CHARS attribute. For a scrollable frame, VIRTUAL-HEIGHT-CHARS specifies the height of the entire frame while HEIGHT-CHARS specifies the height of the visible portion of the frame.

VIRTUAL-HEIGHT-PIXELS attribute

The maximum height of the widget, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a non-scrollable frame, VIRTUAL-HEIGHT-PIXELS has the same value as the HEIGHT-PIXELS attribute. For a scrollable frame, VIRTUAL-HEIGHT-PIXELS specifies the height of the entire frame while HEIGHT-PIXELS specifies the height of the visible portion of the frame.

VIRTUAL-WIDTH-CHARS attribute

The maximum width of the widget, in character units.

Data type: DECIMAL

Access: Readable/WritEable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a non-scrollable frame, VIRTUAL-WIDTH-CHARS has the same value as the WIDTH-CHARS attribute. For a scrollable frame, VIRTUAL-WIDTH-CHARS specifies the width of the entire frame while WIDTH-CHARS specifies the width of the visible portion of the frame.

VIRTUAL-WIDTH-PIXELS attribute

The maximum width of the widget, in pixels.

Data type: INTEGER

Access: Readable/Writeable

Applies to: DIALOG-BOX widget, FRAME widget, WINDOW widget

For a non-scrollable frame, VIRTUAL-WIDTH-PIXELS has the same value as the WIDTH-PIXELS attribute. For a scrollable frame, VIRTUAL-WIDTH-PIXELS specifies the width of the entire frame while WIDTH-PIXELS specifies the width of the visible portion of the frame.

VISIBLE attribute

Indicates whether a widget is currently visible on the display.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DEBUGGER system handle, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is read-only for field groups, menus, menu items, and submenus. A field-level widget must be specified in a frame definition before you set its VISIBLE attribute.

The behavior of the VISIBLE attribute depends on the setting of the HIDDEN attribute of related widgets:

- When you set the VISIBLE attribute of a window to TRUE:
  - Progress displays that window and the widgets it contains whose VISIBLE attributes are already set to TRUE. Otherwise, you must explicitly DISPLAY or VIEW a widget, or otherwise set a widget's VISIBLE attribute to TRUE in order to display it in the window.
  - Progress displays that window and all ancestor windows only if no ancestor window has its HIDDEN attribute set to TRUE. If Progress displays the window, it also displays all descendant windows down to, but not including, the first descendant window whose HIDDEN attribute is set to TRUE.
- When you set the VISIBLE attribute of any widget within a window to TRUE, Progress displays that widget, any ancestor frames, and the window (if necessary), unless the HIDDEN attribute of the window is TRUE. If the window's HIDDEN attribute is TRUE, Progress sets the VISIBLE attributes of the widget and any ancestor frames to TRUE and sets the HIDDEN attributes of the widget and its ancestor frames to FALSE without displaying them.
- When you set the VISIBLE attribute of a frame to TRUE, Progress displays all of its field-level widgets and descendant frames, except those whose HIDDEN attributes are TRUE.

- When you explicitly set the VISIBLE attribute of any widget to TRUE, Progress sets its HIDDEN attribute to FALSE. If you explicitly set the VISIBLE attribute of a field-level widget or child frame to FALSE while its parent frame remains visible, Progress also sets the HIDDEN attribute of the field-level widget or child frame to TRUE. If you explicitly set the VISIBLE attribute of a child window to FALSE, the HIDDEN attribute of the child window remains unchanged, whether or not the parent window is visible.
- The following behavior is true for the browse column:
  - The syntax of the VISIBLE attribute for the browse column is as follows:

```
VISIBLE [ IN BROWSE browse-name ]
```

  - The behavior of the VISIBLE attribute for a browse column does not depend on the setting of the HIDDEN attribute of the related widget.
  - Changing the VISIBLE attribute of a browse column may affect which columns are locked if NUM-LOCKED-COLUMNS has been set. This is because NUM-LOCKED-COLUMNS only applies to visible columns. For example, if the first three columns of a browse are locked and the second column is made not VISIBLE, the fourth column will then become locked.
  - If a widget is not already realized and you set its VISIBLE attribute to TRUE, Progress realizes that widget.
  - In character mode, the VISIBLE attribute is always set to TRUE.

WARNING attribute

Indicates whether the last compilation produced warning messages.

Data type: LOGICAL

Access: Readable

Applies to: COMPILER system handle

If the WARNING attribute is TRUE, there were warning messages from the last compilation.

WHERE-STRING attribute

Returns the current WHERE clause used to link the child of the relation to its parent, beginning with the keyword WHERE but not including the FOR EACH phrase of a prepare statement.

This attribute evaluates to the query string that Progress generates for you based on the relation between parent and child buffers. You can use this attribute to build an extended query of your own based on the default relationship.

Data type: CHARACTER

Access: Readable

Applies to: Data-relation object handle

Widget-Handle property (Windows only; Graphical interfaces only)

The widget handle of the control frame associated with the control-frame COM object.

Return type: WIDGET-HANDLE

Access: Readable

Applies to: CONTROL-FRAME widget, COM object

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.

WIDGET-ENTER attribute

A handle, in a trigger associated with an ENTRY event or a LEAVE event, to the next widget to receive input focus.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: LAST-EVENT system handle

The WIDGET-ENTER attribute is meaningful only within an ENTRY or LEAVE trigger.

For browse widgets, WIDGET-ENTER is different depending on whether the browse is editable or read-only. For editable browse widgets, WIDGET-ENTER contains the widget handle of the column with focus. For read-only browse widgets, WIDGET-ENTER contains the widget handle of the browse.

WIDGET-ID attribute (Windows only; Graphical interfaces only)

An application-defined widget ID for a static or dynamic widget. The value of this attribute must be an even integer value between 2 and 65534, inclusive, and it must be unique across all widget IDs in a window.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FILL-IN widget, FRAME widget, IMAGE widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget

Notes

- Specify an application-defined widget ID when you want to identify the widget at runtime while testing your OpenEdge GUI application with a third-party automated test tool. When specified, Progress uses this application-defined widget ID when creating the widget at runtime, instead of using the widget ID it normally generates by default. The widget ID value of any given widget remains the same across OpenEdge sessions, unless you purposely change the value of the widget ID using this attribute. This allows a third-party automated test tool to identify the same widget consistently each time you run the tool with your application.

To enable application-defined widget IDs in your OpenEdge GUI application, you must specify the Use Widget ID (`-usewidgetid`) startup parameter. For more information about this startup parameter, see *OpenEdge Deployment: Startup Command and Parameter Reference*.

- If you do not specify the Use Widget ID (`-usewidgetid`) startup parameter, and your application contains application-defined widget IDs, Progress ignores any WIDGET-ID attribute or option settings and generates its own widget IDs. In this case, querying the WIDGET-ID attribute returns the Unknown value (`?`).

  If you specify the Use Widget ID (`-usewidgetid`) startup parameter and your application contains application-defined widget IDs, or a combination of application-defined and Progress-defined widget IDs, Progress honors any application-defined widget IDs and assigns Progress-defined widget IDs as necessary. In this case, querying the WIDGET-ID attribute returns the assigned widget ID (whether it be an application-defined value or a Progress-defined value).

- For more information about using application-defined widget IDs when testing OpenEdge GUI applications with third-party automated test tools, see *OpenEdge Development: Programming Interfaces*.

- You can set this attribute only before the widget is realized. Once the widget is realized, this attribute is read-only.

- For frames, Progress uses this attribute value as the basis for assigning a unique widget ID for each child widget within the frame by combining the frame widget ID with the child widget ID. For example, a frame defined with a widget ID of 100 that contains a fill-in widget defined with a widget ID of 2 results in a fill-in widget with a widget ID of 102 at runtime. In this way, the widget ID of each child widget within a given frame is unique within that frame, as well as across multiple instances of that frame within a given window. If the value of the frame widget ID combined with the child widget ID is greater than 65534, Progress displays a warning message and assigns a unique widget ID to the child widget.

  When a frame is displayed as a down frame, you can specify an application-defined widget ID for the first instance of the widget on the down frame. Progress assigns a unique widget ID for each additional instance of the widget on the down frame (based on the number of iterations in the down frame) using consecutive even numbers. Likewise, when a frame contains extent fields, Progress assigns a widget ID to each of the extent elements in the frame.

When a frame is displayed as a dialog box, which is a special type of frame displayed in its own window, the widget ID for any child widgets inside the dialog box must be unique only within that dialog box.

Note: The default widget ID for a frame is 0 (zero). Use caution when defining multiple frames, or multiple instances of a frame, in a single window and allowing the widget IDs to default to 0, because you are more likely to encounter a duplicate widget ID conflict.

- For radio-set widgets, which are built with individual radio buttons, Progress uses this attribute value as the basis for assigning a unique widget ID for each radio button of the given radio-set widget using consecutive even numbers.
- For browse widgets, Progress uses this attribute value as the basis for assigning a unique widget ID for each column within the browse by automatically incrementing the browse widget ID by 1 for each column within the browse sequentially from left to right.
- Progress also provides for assigning widget IDs to widget labels by reserving the previous odd value of each widget ID for the widget's label. For example, if you assign a widget a widget ID of 10, Progress reserves widget ID 9 for the widget's label. Progress does not provide for assigning widget IDs to browse column labels.
- You cannot specify an application-defined widget ID for FIELD-GROUP, LITERAL, MENU, MENU-ITEM, SUB-MENU, or WINDOW widgets.
- If you specify an invalid widget ID value in a static widget definition, Progress generates a compiler error whether the Use Widget ID (`-usewidgetid`) startup parameter is specified or not. If you specify an invalid widget ID value in a dynamic widget definition, Progress generates a runtime error only when the startup parameter is specified.

Caution: To avoid duplicate widget ID conflicts, within and across multiple instances of a widget in a single window, be sure to specify widget IDs within numeric ranges that take other widgets into account. For example, do not specify frame widget IDs in multiples of 10 when you have one or more frame widgets that contain more than 9 child widgets because it will result in a duplicate ID conflict.

WIDGET-LEAVE attribute

A handle, in a trigger associated with an ENTRY event or a LEAVE event, to the widget that had input focus prior to the event.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: LAST-EVENT system handle

The WIDGET-ENTER attribute is meaningful only within an ENTRY or LEAVE trigger.

For browse widgets, WIDGET-LEAVE is different depending on whether the browse is editable or read-only. For editable browse widgets, WIDGET-LEAVE contains the widget handle of the column just left. For read-only browse widgets, WIDGET-LEAVE contains the widget handle of the field-level widget just left.

Width property (Windows only; Graphical interfaces only)

The width of the control-frame and control-frame COM object, in pixels.

Return type: INTEGER

Access: Readable/Writeable

Applies to: CONTROL-FRAME widget, COM object

Setting this value changes the WIDTH-CHARS attribute and WIDTH-PIXELS attribute of the corresponding control-frame widget to an equivalent value.

Note: References to COM object properties and methods extend the syntax used for referencing widget attributes and methods. For more information, see the "Referencing COM object properties and methods" section on page 1501.

WIDTH-CHARS attribute (Graphical interfaces only)

The width of the widget or the display used in the current session, in character units.

Data type: DECIMAL

Access: Readable/Writable

Applies to: BROWSE widget (browse and column), Buffer-field object handle, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SESSION system handle, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The attribute is read-only for field groups, and the SESSION handle.

For control-frames, the WIDTH-CHARS attribute maps to the Width property of the control-frame COM object (ActiveX control container).

For editor widgets, this attribute can set the word wrap margin for the WORD-WRAP attribute. For more information, see the WORD-WRAP attribute reference entry.

For buffer-field objects, the WIDTH-CHARS attribute is the number of characters in the STRING-VALUE, which Progress calculates using the FORMAT attribute. In addition, the WIDTH-CHARS attribute of a buffer-field is readable but not writeable.

For browses, the WIDTH-CHARS attribute sets the width, in characters, of the browse without changing the width of any browse-column. If you change the value of a browse's WIDTH-CHARS or WIDTH-PIXELS attribute, the horizontal scrollbar might appear or disappear, which might cause the number of rows that appear in the viewport to change.

For browse-columns, the WIDTH-CHARS attribute sets the width, in characters, of the browse-column without changing the width of the browse.

WIDTH-PIXELS attribute (Graphical interfaces only)

The width of the widget or the screen display used in the current session, in pixels.

Data type: INTEGER

Access: Readable/Writable

Applies to: BROWSE widget (browse and column), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SESSION system handle, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

The attribute is read-only for field groups, and the SESSION handle.

For control-frames, the WIDTH-PIXELS attribute maps to the Width property of the control-frame COM object (ActiveX control container).

For editor widgets, this attribute can set the word wrap margin for the WORD-WRAP attribute. For more information, see the WORD-WRAP attribute reference entry.

For browses, the WIDTH-PIXELS attribute sets the width, in pixels, of the browse without changing the width of any browse-column. If you change the value of a browse's WIDTH-CHARS or WIDTH-PIXELS attribute, the horizontal scrollbar might appear or disappear, which might cause the number of rows that appear in the viewport to change.

For browse-columns, the WIDTH-PIXELS attribute sets the width, in pixels, of the browse-column without changing the width of the browse.

WINDOW attribute

A handle to the window that owns a widget or that contains the owner of a widget.

Data type: WIDGET-HANDLE

Access: Readable

Applies to: BROWSE widget, BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LITERAL widget, MENU widget, MENU-ITEM widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, SUB-MENU widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

For a menu bar or pop-up menu of a window, the WINDOW and OWNER attributes have the same value.

In the case of a window, the WINDOW attribute returns the window's widget-handle (not its parent's handle, if any). For a menu bar or pop-up menu of a window, the WINDOW and OWNER attributes have the same value.

WINDOW-STATE attribute

The current visual state of a window in the window system.

Data type: INTEGER

Access: Readable/Writeable

Applies to: WINDOW widget

The possible values can be expressed as compiler constants. Table 89 lists these values.

Table 89: Window state values *(1 of 2)*

Compiler constant	Value	Description
WINDOW-MAXIMIZED	1	The window is maximized to fill the entire display.
WINDOW-MINIMIZED	2	The window is minimized (iconified).

Table 89: Window state values *(2 of 2)*

Compiler constant	Value	Description
WINDOW-NORMAL	3	The window is in the "restored" state. Initially, this refers to a state that is neither maximized nor minimized. However, setting WINDOW-STATE to WINDOW-NORMAL restores the window to its previous state, which may be the maximized, minimized, or neither.
WINDOW-DELAYED-MINIMIZE	4	The window is minimized (iconified) the next time a new window, dialog box, or alert box is displayed. This differs from setting WINDOW-STATE to WINDOW-MINIMIZED, which minimizes the window immediately.

You can change the state of a window programmatically by setting the WINDOW-STATE attribute. Note that you can change a window to its maximized state in Windows only.

WINDOW-SYSTEM attribute

A value that indicates the windowing system the application is using.

Data type: CHARACTER

Access: Readable

Applies to: SESSION system handle

Progress sets the value of WINDOW-SYSTEM as follows:

- In Windows 95 and Windows NT, in graphical interfaces:
  - If the Windows 95 shell is running, the value is MS-WIN95.
  - If the Windows 95 shell is not running, the value is MS-WINDOWS.
- In character interfaces, the value is TTY.

Progress supports an override option that enables applications that need the WINDOW-SYSTEM attribute to return the value of MS-WINDOWS for all Microsoft operating systems to do so. To establish this override capability, define the WindowSystem key in the Startup section in the current environment, which might be the registry or an initialization file. If the WindowSystem key is located, the WINDOW-SYSTEM attribute returns the value associated with the WindowSystem key on all platforms.

WORD-WRAP attribute (Graphical interfaces only)

Indicates whether word wrapping is enabled for an editor widget

Data type: LOGICAL

Access: Readable/Writeable

Applies to: EDITOR widget

If WORD-WRAP is TRUE, the editor automatically breaks lines at any word that crosses the word wrap margin of the text area. If WORD-WRAP is FALSE, the editor continues lines beyond the editor border up to the first hard return, and scrolls as required to keep the entered text in view. The user can scroll left and right to view the entire line. The default value for WORD-WRAP is TRUE.

In graphical interfaces, the word wrap margin is set by the WIDTH-CHARS, WIDTH-PIXELS, or INNER-CHARS attribute. In character interfaces, the word wrap margin is determined by either the WIDTH-CHARS or BUFFER-CHARS attribute, whichever is larger.

In Windows, both the regular editor and the large editor support WORD-WRAP.

Note: If the SCROLLBAR-HORIZONTAL attribute is set to TRUE, then WORD-WRAP is automatically set to FALSE. Likewise, if you set the WORD-WRAP attribute to TRUE, then SCROLLBAR-HORIZONTAL is automatically set to FALSE.

You can set this attribute only before the widget is realized.

WORK-AREA-HEIGHT-PIXELS attribute

Indicates the height of the work-area in pixels. The work-area is the portion of the Windows desktop that is not hidden by task bars. That is, the dimensions of the work-area are the dimensions of the Windows desktop minus the dimensions of all task bars on the Windows desktop.

Data type: INTEGER

Access: Readable

Applies to: SESSION system handle

In character interfaces, this attribute returns the Unknown value (?).

WORK-AREA-WIDTH-PIXELS attribute

Indicates the width of the work-area in pixels. The work-area is the portion of the Windows desktop that is not hidden by task bars. That is, the dimensions of the work-area are the dimensions of the Windows desktop minus the dimensions of all task bars on the Windows desktop.

Data type: INTEGER

Access: Readable

Applies to: SESSION system handle

On character platforms, this attribute returns the Unknown value (?).

WORK-AREA-X attribute

The starting x-coordinate (the upper left-hand corner) of the work-area in pixels. The work-area is the portion of the Windows desktop that is not hidden by task bars. That is, the dimensions of the work-area are the dimensions of the Windows desktop minus the dimensions of all task bars on the Windows desktop.

Data type: INTEGER

Access: Readable

Applies to: SESSION system handle

On character platforms, this attribute returns the Unknown value (?).

WORK-AREA-Y attribute

The starting y-coordinate (the upper left-hand corner) of the work-area in pixels. The work-area is the portion of the Windows desktop that is not hidden by task bars. That is, the dimensions of the work-area are the dimensions of the Windows desktop minus the dimensions of all task bars on the Windows desktop.

Data type: INTEGER

Access: Readable

Applies to: SESSION system handle

On character platforms, this attribute returns the Unknown value (?).

WRITE( ) method

Writes data to the socket.

Return type: LOGICAL

Applies to: Socket object handle

Syntax

```
WRITE( buffer , position , bytes-to-write )
```

buffer

A MEMPTR expression which contains data which should be written to the socket.

position

An INTEGER expression greater than 0 that indicates the starting byte position within *buffer* which should be written to the socket.

bytes-to-write

An INTEGER expression that specifies the number of bytes to be written to the socket.

WRITE( ) returns TRUE if the write operation succeeded normally and returns FALSE otherwise. An error can occur if:

- The position parameter is not greater than 0.
- Amount of information requested to write exceeds the amount of data in the buffer.
- Writing to the socket fails.

This method expects *buffer* to identify a MEMPTR variable which already has a region of memory associated with it. The developer must call the SET-SIZE statement to allocate memory and associate it with a MEMPTR variable. It is the responsibility of the developer to free this memory, also via the SET-SIZE statement. The WRITE method will fail if the size of *buffer* is less than *bytes-to-write*.

Note Even if the WRITE( ) method returns TRUE, not all the bytes may have actually been written. To find out how many bytes were written, check the BYTES-WRITTEN attribute.

WRITE-CDATA( ) method

Adds a CDATA block to an XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-CDATA( value )
```

value

A LONGCHAR expression evaluating to the value of the CDATA block.

Call this method to add character data to the XML document. Character data in an XML document belongs exclusively in leaf nodes. (A leaf node is a bottom node; one that does not have any child nodes in a hierarchical tree structure, like an XML document.) Character data cannot appear outside of the root (document) node. The SAX-writer puts the block into the format of a CDATA section by adding the correct open and close tags. For example:

```
<![CDATA[ your_CDATA_block ]]>
```

This method does not change the WRITE-STATUS attribute.

Note: The CHARACTER data is serialized and not escaped by the SAX-writer.

WRITE-CHARACTERS( ) method

Adds character data to an XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-CHARACTERS( { chardata | longchar } )
```

chardata

An expression that evaluates to a CHARACTER variable that contains the XML text.

longchar

An expression that evaluates to a LONGCHAR variable that contains the XML text.

Call this method to add character data to the XML document. Character data in an XML document belongs exclusively in leaf nodes. (A leaf node is a bottom node; one that does not have any child nodes in a hierarchical tree structure, like an XML document.) Character data cannot appear outside of the root (document) node.

This method sets the status to SAX-WRITE-CONTENT.

The method escapes all special characters according to the XML specification. For example, "<" is changed to "*<*".

If the STRICT attribute is TRUE and the call would result in CHARACTER data being written at the document level (that is, outside of the root node), then the method fails.

WRITE-COMMENT( ) method

Adds a comment to the XML document represented by a SAX-writer object.

Data type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-COMMENT( value )
```

value

A LONGCHAR expression evaluating to the text of the comment.

Call this method to add a comment node to the XML document. You can add comments at any time. The SAX-writer object creates the comment by enclosing the CHARACTER expression in open and close comment markers (<!-- and --!>). For example:

```
<!-- This is a comment --!>
```

WRITE-DATA-ELEMENT( ) method

Creates a complete XML node in a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-DATA-ELEMENT( name, { chardata | longchar } [, namespace-URI ] )
```

name

A LONGCHAR expression evaluating to the fully qualified or unqualified name of the element.

chardata

An expression that evaluates to a CHARACTER variable that contains the XML text.

longchar

An expression that evaluates to a LONGCHAR variable that contains the XML text.

namespace-URI

A CHARACTER expression evaluating to the URI of the element, or an empty string ("") or the Unknown value (?).

Creates a complete XML node. This method call sets the WRITE-STATUS to SAX-WRITE-ELEMENT.

If you use *namespace-URI*, then the prefix will be resolved in the following order:

1. The method attempts to extract the namespace from the name.
2. The method attempts to extract the namespace from a previously declared namespace.
3. The method attempts to generate the default namespace.

If *name* contains a prefix and *namespace-URI* is used, and this call is the first instance of the *namespace-URI,* then the namespace will be added to the element. This is equivalent to calling the DECLARE-NAMESPACE method.

If only the *name* is used and it contains a prefix, then the SAX-writer attempts to resolve the prefix to a namespace.

This technique is logically equivalent to calling the START-ELEMENT, WRITE-CHARACTERS, and END-ELEMENT methods where *name* and *namespace-URI* are the parameters of START-ELEMENT and END-ELEMENT, and *chardata* is the parameter of WRITE-CHARACTERS.

Note that attributes and namespaces cannot be added after you call this method. If you need to add either to the element, then use the START-ELEMENT method.

If the STRICT attribute is TRUE, the FRAGMENT attribute is FALSE, and the invocation would result in more than one document-level element (that is, root node), then the method fails. Also, if STRICT is TRUE, an external DTD has been declared, and the invocation would create the root node, the *name* used for the DTD declaration must match the *name* of the root node or the method fails.

See also DECLARE-NAMESPACE( ) method

WRITE-EMPTY-ELEMENT( ) method

Creates an empty XML node in a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-EMPTY-ELEMENT( name [, namespace-URI ] )
```

name

A LONGCHAR expression evaluating to the fully qualified or unqualified name of the element.

namespace-URI

A CHARACTER expression evaluating to the URI of the element, or an empty string ("") or the Unknown value (?) if the element doesn't contain a namespace.

Creates an empty XML node. This method call sets the WRITE-STATUS to SAX-WRITE-TAG.

If *namespace-URI* is present, then the prefix will be resolved in the following order:

1. The method attempts to extract the namespace from the *name*.
2. The method attempts to extract the namespace from a previously declared namespace.
3. The method attempts to generate the default namespace.

If the *name* contains a prefix, *namespace-URI* is present, and this is the first instance of the *namespace-URI* then the namespace is added to the element. This is equivalent to calling the DECLARE-NAMESPACE method.

If only the *name* is present and it contains a prefix, then the SAX-writer attempts to resolve the prefix to a namespace.

Although this method call appears to be logically equivalent to a START-ELEMENT invocation directly followed by its corresponding END-ELEMENT invocation, the two techniques produce different outputs. The START-ELEMENT and END-ELEMENT methods produce a pair of tags; `<element></element>`. WRITE-EMPTY-ELEMENT produces the empty element tag; *<element/>*.

If STRICT is TRUE, FRAGMENT is FALSE, and the invocation would result in more than one document-level element, (that is, root node), then the method fails.

WRITE-ENTITY-REF( ) method

Adds an entity reference to the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-ENTITY-REF ( value )
```

value

A LONGCHAR expression evaluating to the value of the entity reference.

Call this method to add an entity reference to the XML document. You can add entity references at any time during the write.

This method does not change the WRITE-STATUS attribute.

You cannot add entity references using the WRITE-CHARACTERS method because the entity references contain the escapable character “&”. For example, if you add the entity reference `fromname` with the WRITE-CHARACTERS method, this call, `xmlwh:WRITE-CHARACTERS("&fromname;")`, produces the invalid value `&fromname;` in the XML document. However, you can add entity references using the WRITE-FRAGMENT method, since it does not escape special characters.

Do not include the special characters when inserting the reference, only the reference value. For example, `xmlwh:entity-reference("fromname")` produces `&fromname;` in the XML document.

See also WRITE-CHARACTERS( ) method, WRITE-FRAGMENT( ) method

WRITE-EXTERNAL-DTD( ) method

Adds an external Document Type Definition (DTD) reference to an XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-EXTERNAL-DTD( name, systemID [ , publicID ] )
```

name

A LONGCHAR expression evaluating to the fully qualified or unqualified name of the XML document root node.

systemID

A LONGCHAR expression evaluating to the system ID of the DTD.

publicID

A LONGCHAR expression evaluating to the public ID of the DTD.

Call this method to add an external DTD reference to the prolog of the XML document.

You can only call this method before the first call of START-ELEMENT. That is, only call this method when the WRITE-STATUS is SAX-WRITE-BEGIN. After the call, the status remains SAX-WRITE-BEGIN.

If the STRICT attribute is TRUE, and you call this method after you create the root element (or when the CREATE-FRAGMENT attribute is TRUE), then the method fails.

The value of *name* must match the value in the root node. If the STRICT attribute is TRUE and the two values do not match, then the method fails and generates an error.

WRITE-FRAGMENT( ) method

Adds character data to the XML document represented by a SAX-writer object.

Return type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-FRAGMENT( { chardata | longchar | x-noderef } )
```

chardata

An expression that evaluates to a CHARACTER variable that contains the XML text.

longchar

An expression that evaluates to a LONGCHAR variable that contains the XML text.

noderef

A valid X-NODEREF handle that contains the XML text.

Call this method to add un-escaped CHARACTER data to the XML document. This allows the adding of XML fragments to the document without the special characters being escaped to their XML representation. For example, "<" escapes to `<`. It is up to the developer to ensure that the characters written are proper XML with the correct characters escaped. Even if the STRICT attribute is TRUE, the SAX-writer will not validate what is written.

You can call this method at any time during the write. This method changes the WRITE-STATUS attribute to SAX-WRITE-CONTENT.

WRITE-MESSAGE( ) method

Writes a user message to the current log file.

For an interactive or batch client, the WRITE-MESSAGE( ) method writes the log entries to the log file specified by the LOGFILE-NAME attribute or the Client Logging (`-clientlog`) startup parameter. For WebSpeed agents and AppServer servers, the WRITE-MESSAGE( ) method writes the log entries to the server log file.

Return type: LOGICAL

Applies to: LOG-MANAGER system handle

Syntax

```
WRITE-MESSAGE( msg-exp [, subsys-expr ])
```

msg-exp

A character expression or variable representing the message to write to the log file.

subsys-expr

A character expression representing the subsystem identifier to write to the log file. The default is "APPL". You can provide your own subsystem identifier. The subsystem identifier has a character limit 10 characters, and is padded to 10 characters. If you provide a subsystem identifier longer than 10 characters, WRITE-MESSAGE( ) writes only the first 10 characters.

Example

The following is an example:

```
LOG-MANAGER:WRITE-MESSAGE("Got here, x=" + string(x), "DEBUG1").
```

The following line appears in the log file:

```
[04/12/05@13:19:19.742-0500] P-003616 T-001984 1 4GL DEBUG1    Got here, x=5
```

Notes

- If the WRITE-MESSAGE( ) method succeeds, it returns TRUE. If it fails, it returns FALSE.
- If there is no client log file, the WRITE-MESSAGE( ) method returns FALSE and displays a warning message indicating this operation is not valid when there is no log file. For an interactive or batch client, the WRITE-MESSAGE( ) method writes the warning message to the current output device.
- When the client writes messages using the WRITE-MESSAGE( ) method, the component identifier in the message header is the default component identifier for the client executable writing to the log. For example, the component identifier for a GUI or character client is “4GL”, for WebSpeed is “WS”, and for AppServer is “AS”. You can provide your own subsystem identifier. The default is "APPL". The subsystem identifier has a character limit of 10 characters, and is padded to 10 characters.

WRITE-PROCESSING-INSTRUCTION( ) method

Creates a processing instruction node in an XML document represented by a SAX-writer object.

Return Type: LOGICAL

Applies to: SAX-writer object handle

Syntax

```
WRITE-PROCESSING-INSTRUCTION( target, data )
```

target

A LONGCHAR expression evaluating to the target of the processing instruction.

data

A LONGCHAR expression that evaluates to the data associated with the processing instruction.

Call this method to add a processing instruction node to the XML document. You can add processing instructions at any time. The SAX-writer object creates the processing instruction by enclosing the CHARACTER expression in open and close processing instruction markers (**<?** and **?>**). For example:

```
<?xml version="1.0"?>
```

This method does not change the WRITE-STATUS attribute.

WRITE-STATUS attribute

The current state of a XML write in a SAX-writer object.

Data type: INTEGER

Access: Readable

Applies to: SAX-writer object handle

The default value is SAX-WRITE-IDLE.

The possible values for WRITE-STATUS can assume are shown in Table 90.

Table 90: WRITE-STATUS attribute values *(1 of 2)*

WRITE-STATUS value	Description
SAX-WRITE-IDLE	No writing has occurred.
SAX-WRITE-BEGIN	The START-DOCUMENT method has been called and writing has begun.
SAX-WRITE-TAG	The writer has written an opening tag. This is the only time that attributes can be inserted with INSERT-ATTRIBUTE and DECLARE-NAMESPACE.
SAX-WRITE-ELEMENT	The writer is within an element.
SAX-WRITE-CONTENT	The writer has written the content of an element. In other words, the WRITE-CHARACTERS method has been called.

Table 90: WRITE-STATUS attribute values *(2 of 2)*

WRITE-STATUS value	Description
SAX-WRITE-COMPLETE	The END-DOCUMENT method has been called and writing is complete.
SAX-WRITE-ERROR	The SAX-writer could not start or could not continue. Likely causes include: SAX-writer could not be loaded, the XML target could not be written to, a method call fails, etc. This is the status if there is an invalid XML generated while STRICT is TRUE. If the status is SAX-WRITE-ERROR then no attributes can be written and the only method that can be called is RESET.

WRITE-XML( ) method

Writes an XML document from a ProDataSet, temp-table, or temp-table buffer object. You can write the XML representation of the object with data, schema, or both. If you include schema, it is written using the XML Schema Definition (XSD) language.

When writing data from a ProDataSet object, Progress writes the current version of data in each row of each table in the ProDataSet object. However, you can also include any before-image data, so that both the current and original versions of the data in each table row are written.

When writing schema for a ProDataSet object, Progress writes all table definitions as well as relation and index definitions. When writing schema for a temp-table or temp-table buffer object, Progress writes only table and index definitions.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
WRITE-XML ( target-type, { file | stream | memptr | handle | longchar }
[, formatted [, encoding [, schema-location [, write-xmlschema
[, min-xmlschema [, write-before-image ] ] ] ] ] ] )
```

target-type

A CHARACTER expression that specifies the target XML document type. Valid values are: “FILE”, “STREAM”, “MEMPTR”, “HANDLE”, and “LONGCHAR”.

file

A CHARACTER expression that specifies the name of a file to which Progress writes the XML document text. You can specify an absolute pathname or a relative pathname (based on the current working directory). If a file with the specified name already exists, Progress verifies that the file is writeable and overwrites the file.

stream

A CHARACTER expression that specifies the name of a stream. If you specify the empty string (""), Progress writes the XML document text to the default unnamed output stream. For WebSpeed, write the XML document text to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using Progress unnamed output streams, see the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces*. For more information about using WebSpeed-defined output streams, see *OpenEdge Application Server: Developing WebSpeed Applications*.

memptr

A MEMPTR variable to contain the XML document text in memory. The method allocates the required amount of memory for the XML document text and sets the size of the variable. When you are finished using the MEMPTR, you must free the associated memory by executing SET-SIZE(*memptr*) = 0 on the MEMPTR.

handle

An X-document object handle or X-noderef object handle. If the specified handle contains XML text, Progress deletes the existing text first.

longchar

A LONGCHAR variable to contain the XML document text in memory.

Progress saves the XML document text to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the *encoding* option. If you do not specify a character encoding for the XML document text, Progress saves the LONGCHAR variable in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE function) and the fixed code page is not equivalent to the character encoding you specify in the *encoding* option, the WRITE-XML( ) method returns an error and the XML document is not saved to the LONGCHAR.

formatted

An optional LOGICAL expression where TRUE directs Progress to format the XML document text in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

An optional CHARACTER expression that specifies the name of the character encoding Progress uses to write the XML document text. The default encoding is UTF-8.

The encoding name must be an Internet Assigned Numbers Authority (IANA) name supported by the Progress XML Parser. For a list of supported IANA encodings and their corresponding Progress code pages, see Table 64 in the ENCODING attribute reference entry in this book.

Note: Progress records this character encoding in the encoding declaration in the XML document's prologue. If you specify the empty string ("") or the Unknown value (?), Progress uses the default encoding of UTF-8. In this case, Progress does not record the character encoding in the XML document's encoding declaration.

If *target-type* is HANDLE, the X-document's ENCODING attribute is also set.

schema-location

An optional CHARACTER expression that specifies the name of an external XML Schema file. The method uses this value to set the `xsi:schemaLocation` or `xsi:noNamespaceSchemaLocation` attribute in the XML document. If the ProDataSet or temp-table object's NAMESPACE-URI attribute is the empty string ("") or the Unknown value (?), the method adds the `xsi:noNamespaceSchemaLocation` attribute in the XML document and sets it to this value. If the NAMESPACE-URI attribute is not the empty string ("") or the Unknown value (?), the method adds the `xsi:schemaLocation` attribute to the XML document with a value of "`namespace-uri<space>`*`schema-location`*". The default value is the Unknown value (?).

Note: You must provide the location of an actual XML Schema file. Consider using the WRITE-XMLSCHEMA( ) method to generate the XML Schema file.

If you specify *write-xmlschema* as TRUE, you cannot specify *schema-location*.

write-xmlschema

An optional LOGICAL expression where TRUE directs Progress to write the ProDataSet or temp-table object's relational structure as in-line XML Schema along with the data, and FALSE directs Progress to write only the data. The default value is FALSE.

If you specify TRUE, you cannot specify *schema-location*. If you specify FALSE, you must also specify *min-xmlschema* as FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

Note: If you specify TRUE and the NAMESPACE-URI attribute value for a temp-table buffer within a ProDataSet object is different than that of the ProDataSet object, the method creates a separate XML Schema file for the temp-table definition. The namespace URI for the temp-table is imported into the ProDataSet schema, with a `schemaLocation` pointing to a separate XML Schema file containing the temp-table definition. Multiple namespaces are supported only when *target-type* is "FILE". If the ProDataSet object contains multiple namespaces and *target-type* is not "FILE", the method generates an error and returns FALSE.

min-xmlschema

An optional LOGICAL expression where TRUE directs Progress to write the minimum amount of schema when it writes the XML Schema representation of the object, and FALSE directs Progress to write the complete schema including Progress-specific schema attributes. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When TRUE, Progress-specific schema information (such as, field format, non-unique indexes, and so on) is omitted from the XML Schema. If the Progress data type of the temp-table field is not the default Progress data type for the XML Schema type, Progress writes the `prodata:`*`dataType`* XML Schema attribute for the field. If the initial value of the temp-table field is TODAY, NOW, or UNKNOWN (and UNKNOWN is not the default initial value for the field's data type), Progress writes the `prodata:`*`initial`* XML Schema attribute for the field.

If you specify *write-xmlschema* as FALSE, you must also specify *min-xmlschema* as FALSE.

write-before-image

An optional LOGICAL expression where TRUE directs Progress to write any before-image table data and error information in addition to the ProDataSet object data, and FALSE directs Progress to write only the ProDataSet object data. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

Examples The following code example defines a static ProDataSet object, attaches its data sources, fills the ProDataSet object, and writes the ProDataSet object to an XML document in a nested manner:

```
DEFINE TEMP-TABLE ttCust LIKE customer.
DEFINE TEMP-TABLE ttOrd  LIKE order.
DEFINE TEMP-TABLE ttInv  LIKE invoice.

DEFINE DATASET DSET FOR ttCust, ttOrd, ttInv
  DATA-RELATION CustOrd FOR ttCust,
    ttOrd RELATION-FIELDS(cust-num,cust-num) NESTED
  DATA-RELATION OrdInv FOR ttOrd,
    ttInv RELATION-FIELDS(order-num,order-num) NESTED.

DEFINE DATA-SOURCE dsCust FOR customer.
DEFINE DATA-SOURCE dsOrd  FOR order.
DEFINE DATA-SOURCE dsInv  FOR invoice.

BUFFER ttCust:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsCust:HANDLE).
BUFFER ttOrd:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsOrd:HANDLE).
BUFFER ttInv:HANDLE:ATTACH-DATA-SOURCE(DATA-SOURCE dsInv:HANDLE).

DATA-SOURCE dsCust:FILL-WHERE-STRING = "where cust-num = 2 ".
DATASET DSET:FILL().

DEFINE VARIABLE cTargetType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted AS LOGICAL NO-UNDO.
DEFINE VARIABLE cEncoding AS CHARACTER NO-UNDO.
DEFINE VARIABLE cSchemaLocation AS CHARACTER NO-UNDO.
DEFINE VARIABLE lWriteSchema AS LOGICAL NO-UNDO.
DEFINE VARIABLE lMinSchema AS LOGICAL NO-UNDO.
DEFINE VARIABLE retOK AS LOGICAL NO-UNDO.

ASSIGN
     cTargetType = "file"
     cFile = "dset.xml"
     lFormatted = YES
     cEncoding = ?
     cSchemaLocation = ?
     lWriteSchema = NO
     lMinSchema = NO.

retOK = DATASET DSET:WRITE-XML(cTargetType,
                               cFile,
                               lFormatted,
                               cEncoding,
                               cSchemaLocation,
                               lWriteSchema,
                               lMinSchema).
```

The following code example defines a static temp-table object, populates the temp-table object (code not shown), and writes the temp-table object to an XML document:

```
DEFINE TEMP-TABLE ttCust LIKE Customer.
DEFINE VARIABLE cTargetType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted AS LOGICAL NO-UNDO.
DEFINE VARIABLE cEncoding AS CHARACTER NO-UNDO.
DEFINE VARIABLE cSchemaLocation AS CHARACTER NO-UNDO.
DEFINE VARIABLE lWriteSchema AS LOGICAL NO-UNDO.
DEFINE VARIABLE lMinSchema AS LOGICAL NO-UNDO.
DEFINE VARIABLE retOK AS LOGICAL NO-UNDO.

/*  code to populate the temp-table  */

ASSIGN
     cTargetType = "file"
     cFile = "ttCust.xml"
     lFormatted = YES
     cEncoding = ?
     cSchemaLocation = ?
     lWriteSchema = NO
     lMinSchema = NO.

retOK = TEMP-TABLE ttCust:WRITE-XML(cTargetType,
                                    cFile,lFormatted,
                                    cEncoding,
                                    cSchemaLocation,
                                    lWriteSchema,
                                    lMinSchema).
```

Notes

- You can specify how a temp-table column is represented in XML (that is, as an ELEMENT, ATTRIBUTE, or TEXT) by:
  - Setting the XML-NODE-TYPE attribute on the Buffer-field object handle.
  - Specifying the XML-NODE-TYPE option on the DEFINE TEMP-TABLE statement.
- When writing data from a ProDataSet object that contains data-relations, you can nest child rows of a ProDataSet buffer within their parent rows in the resulting XML document by:
  - Setting the NESTED attribute on the Data-relation object handle to TRUE.
  - Specifying the NESTED option for the data-relation on the DEFINE DATASET statement.
  - Specifying the NESTED option in the ADD-RELATION( ) method.
- If your temp-tables contain array fields, third party products utilizing the XML might not map the Progress array field to an array column or object. For best interoperability with third party products, flatten array fields into individual fields.
- You cannot write an XML document from a database buffer.

See also

ENCODING attribute, FIX-CODEPAGE function, NAMESPACE-PREFIX attribute, NAMESPACE-URI attribute, NESTED attribute, READ-XML( ) method, READ-XMLSCHEMA( ) method, WRITE-XMLSCHEMA( ) method, XML-NODE-TYPE attribute

WRITE-XMLSCHEMA( ) method

Writes an XML representation of the schema for a ProDataSet, temp-table, or temp-table buffer object (that is, an XML Schema file). The XML Schema is written using the XML Schema Definition (XSD) language.

When writing schema for a ProDataSet object, Progress writes all table definitions as well as relation and index definitions. When writing schema for a temp-table or temp-table buffer object, Progress writes only table and index definitions.

Return type: LOGICAL

Applies to: Buffer object handle, ProDataSet object handle, Temp-table object handle

Syntax

```
WRITE-XMLSCHEMA ( target-type,
   { file | stream | memptr | handle | longchar }
   [, formatted [, encoding [, min-xmlschema ] ] ] )
```

target-type

A CHARACTER expression that specifies the target XML Schema document type. Valid values are: “FILE”, “STREAM”, “MEMPTR”, “HANDLE”, and “LONGCHAR”.

file

A CHARACTER expression that specifies the name of a file to which Progress writes the XML Schema document text. You can specify an absolute pathname or a relative pathname (based on the current working directory). If a file with the specified name already exists, Progress verifies that the file is writeable and overwrites the file.

stream

A CHARACTER expression that specifies the name of a stream. If you specify the empty string (""), Progress writes the XML Schema document text to the default unnamed output stream. For WebSpeed, write the XML Schema document text to the WebSpeed-defined output stream (WEBSTREAM).

For more information about using Progress unnamed output streams, see the DEFINE STREAM statement reference entry in this book and the chapter on alternate I/O sources in *OpenEdge Development: Programming Interfaces*. For more information about using WebSpeed-defined output streams, see *OpenEdge Application Server: Developing WebSpeed Applications*.

memptr

A MEMPTR variable to contain the XML Schema document text in memory. The method allocates the required amount of memory for the XML document text and sets the size of the variable. When you are finished using the MEMPTR, you must free the associated memory by executing SET-SIZE(*memptr*) = 0 on the MEMPTR.

handle

An X-document object handle or X-noderef object handle. If the specified handle contains XML text, Progress deletes the existing text first.

longchar

A LONGCHAR variable to contain the XML Schema document text in memory.

Progress saves the XML Schema document text to the LONGCHAR variable in the code page that corresponds to the character encoding you specify in the *encoding* option. If you do not specify a character encoding for the XML Schema document text, Progress saves the LONGCHAR variable in UTF-8.

If the LONGCHAR variable's code page is fixed (that is, set using the FIX-CODEPAGE function) and the fixed code page is not equivalent to the character encoding you specify in the *encoding* option, the WRITE-XMLSCHEMA( ) method returns an error and the XML Schema document is not saved to the LONGCHAR.

formatted

An optional LOGICAL expression where TRUE directs Progress to format the XML Schema document text in a hierarchical manner using extra white space, carriage returns, and line feeds. The default value is FALSE.

If you specify the Unknown value (?), the method uses the default value of FALSE.

encoding

An optional CHARACTER expression that specifies the name of the character encoding Progress uses to write the XML Schema document text. The default encoding is UTF-8.

The encoding name must be an Internet Assigned Numbers Authority (IANA) name supported by the Progress XML Parser. For a list of supported IANA encodings and their corresponding Progress code pages, see Table 64 in the ENCODING attribute reference entry in this book.

Note: Progress records this character encoding in the encoding declaration in the XML document's prologue. If you specify the empty string ("") or the Unknown value (?), Progress uses the default encoding of UTF-8. In this case, Progress does not record the character encoding in the XML document's encoding declaration.

If *target-type* is HANDLE, the X-document's ENCODING attribute is also set.

min-xmlschema

An optional LOGICAL expression where TRUE directs Progress to write the minimum amount of schema for the object, and FALSE directs Progress to write the complete schema including Progress-specific schema attributes. The default value is FALSE. If you specify the Unknown value (?), the method uses the default value of FALSE.

When TRUE, Progress-specific schema information (such as, field format, non-unique indexes, and so on) is omitted from the XML Schema. If the Progress data type of the temp-table field is not the default Progress data type for the XML Schema type, Progress writes the `prodata:`*`dataType`* XML Schema attribute for the field. If the initial value of the temp-table field is TODAY, NOW, or UNKNOWN (and UNKNOWN is not the default initial value for the field's data type), Progress writes the `prodata:`*`initial`* XML Schema attribute for the field.

Examples The following code example defines a static ProDataSet object and writes the ProDataSet object schema to an XML Schema file:

```
DEFINE TEMP-TABLE ttCust LIKE customer.
DEFINE TEMP-TABLE ttOrd  LIKE order.
DEFINE TEMP-TABLE ttInv  LIKE invoice.

DEFINE DATASET DSET FOR ttCust, ttOrd, ttInv
  DATA-RELATION CustOrd FOR ttCust,
    ttOrd RELATION-FIELDS(cust-num,cust-num) NESTED
  DATA-RELATION OrdInv FOR ttOrd,
    ttInv RELATION-FIELDS(order-num,order-num) NESTED.

DEFINE VARIABLE cTargetType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted AS LOGICAL NO-UNDO.
DEFINE VARIABLE cEncoding AS CHARACTER NO-UNDO.
DEFINE VARIABLE lMinSchema AS LOGICAL NO-UNDO.
DEFINE VARIABLE retOK AS LOGICAL NO-UNDO.

ASSIGN
     cTargetType = "file"
     cFile = "cust-ord-inv.xsd"
     lFormatted = YES
     cEncoding = ?
     lMinSchema = NO.

retOK = DATASET DSET:WRITE-XMLSCHEMA(cTargetType,
                                     cFile,
                                     lFormatted,
                                     cEncoding,
                                     lMinSchema).
```

The following code example defines a static temp-table object, and writes the temp-table object schema to an XML Schema file:

```
DEFINE TEMP-TABLE ttCust LIKE Customer.
DEFINE VARIABLE cTargetType AS CHARACTER NO-UNDO.
DEFINE VARIABLE cFile AS CHARACTER NO-UNDO.
DEFINE VARIABLE lFormatted AS LOGICAL NO-UNDO.
DEFINE VARIABLE cEncoding AS CHARACTER NO-UNDO.
DEFINE VARIABLE lMinSchema AS LOGICAL NO-UNDO.
DEFINE VARIABLE retOK AS LOGICAL NO-UNDO.

ASSIGN
     cTargetType = "file"
     cFile = "ttCust.xsd"
     lFormatted = YES
     cEncoding = ?
     lMinSchema = NO.

retOK = TEMP-TABLE ttCust:WRITE-XMLSCHEMA(cTargetType,
                                          cFile,
                                          lFormatted,
                                          cEncoding,
                                          lMinSchema).
```

Notes

- If the NAMESPACE-URI attribute value for a temp-table within a ProDataSet object is different than that of the ProDataSet object, the method creates a separate XML Schema file for the temp-table definition. The namespace URI for the temp-table is imported into the ProDataSet schema, with a `schemaLocation` pointing to a separate XML Schema file containing the temp-table definition. Multiple namespaces are supported only when *`target-type`* is “FILE”. If you specify multiple namespaces and *`target-type`* is not “FILE”, the method generates an error and returns FALSE.
- You can specify how a temp-table column is represented in XML Schema (that is, as an ELEMENT, ATTRIBUTE, or TEXT) by:
  - Setting the XML-NODE-TYPE attribute on the Buffer-field object handle.
  - Specifying the XML-NODE-TYPE option on the DEFINE TEMP-TABLE statement.

- When writing schema for a ProDataSet object that contains data-relations, you can nest child rows of a ProDataSet buffer within their parent rows in the resulting XML document by:
  - Setting the NESTED attribute on the Data-relation object handle to TRUE.
  - Specifying the NESTED option for the data-relation on the DEFINE DATASET statement.
  - Specifying the NESTED option in the ADD-RELATION( ) method.
- If your temp-tables contain array fields, third party products utilizing the XML Schema might not map the Progress array field to an array column or object. For best interoperability with third party products, flatten array fields into individual fields.
- You cannot write an XML representation of the schema for a database buffer.

See also ENCODING attribute, FIX-CODEPAGE function, NAMESPACE-PREFIX attribute, NAMESPACE-URI attribute, NESTED attribute, READ-XML( ) method, READ-XMLSCHEMA( ) method, WRITE-XML( ) method, XML-NODE-TYPE attribute

X attribute

The pixel location of the left edge of a widget relative to the left edge of the parent widget or the display. The pixel location of the mouse cursor relative to the left edge of the display (for the last mouse event).

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is read-only for field groups, browse cells, and the LAST-EVENT handle.

For all user interface widgets except windows, the X attribute specifies the location, in pixels, of the left edge of the widget relative to the left edge of its parent widget. In windows, it is the location of the left edge of the window relative to the left edge of the display.

For a browse column, the X attribute returns the Unknown value (?) if the column is hidden.

For control-frames, the X attribute maps to the Left property of the control-frame COM object (ActiveX control container).

For the LAST-EVENT handle, the X attribute returns the pixel location of a mouse event relative to the left edge of the current frame.

This attribute is functionally equivalent to the COLUMN attribute.

X-DOCUMENT attribute

Contains the X-document object handle of an XML document posted to the transaction server or the Unknown value (?) if there isn't one.

Data type: HANDLE

Access: Readable

Applies to: WEB-CONTEXT system handle

XML-DATA-TYPE attribute

Returns the XML Schema data type for the buffer-field object.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

The XML Schema data type must be compatible with the Progress data type for the field.

For more information about the Progress XML data type mapping rules, see *OpenEdge Development: Programming Interfaces*.

If the temp-table schema was created from an XML Schema, this attribute is the same as the `xsd:type` attribute in the XML Schema.

XML-NODE-TYPE attribute

The XML node type of the buffer-field object, which lets you specify how a buffer field is represented in XML and XML Schema. Valid XML node types are: "ATTRIBUTE", "ELEMENT", "HIDDEN", and "TEXT". The default value is "ELEMENT".

Data type: CHARACTER

Access: Readable/Writeable

Applies to: Buffer-field object handle

Table 91 lists the valid XML node types.

Table 91: XML node types

When the XML node type is ...	The buffer field is ...
ATTRIBUTE	Represented as an attribute of the temp-table element in both the XML Schema and data.
ELEMENT	Represented as a child element of the temp-table element in both the XML Schema and data.
HIDDEN	Omitted from both the XML Schema and data.
TEXT	Represented as a text element in both the XML Schema and data. **Note:** Each table can contain only one TEXT field. When a table contains a TEXT field, it cannot contain ELEMENT fields; it can contain only ATTRIBUTE fields. A table that contains a TEXT field cannot be part of a nested data-relation.

The XML node type of a buffer field that represents an array must be either "ELEMENT" or "HIDDEN".

See also WRITE-XML( ) method, WRITE-XMLSCHEMA( ) method

XML-SCHEMA-PATH attribute (WebSpeed Only)

A delimiter-separated list of directory paths for the XML Document Type Definition (DTD) associated with a particular XML document in a WebSpeed environment. Searched if the XML document contains a relative path to the DTD.

Data type: CHARACTER

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

Almost identical to the SCHEMA-PATH attribute of the X-document and SAX-reader objects. For more information on SCHEMA-PATH, see the reference entry for the SCHEMA-PATH attribute.

What XML-SCHEMA-PATH avoids

In WebSpeed, the first time you access the X-document handle or any of its attributes, you trigger a load of the document, which precedes your access. For example, if you set the SCHEMA-PATH attribute of X-document (of WEB-CONTEXT) before accessing X-document (of WEB-CONTEXT) or any of its attributes, when the document arrives, WebSpeed first loads the document, then sets SCHEMA-PATH to your value. So at load time, SCHEMA-PATH might not contain your value, which might cause WebSpeed not to find your DTD, which might cause validation of the document to fail.

By contrast, if you set XML-SCHEMA-PATH instead of SCHEMA-PATH, when the document arrives, WebSpeed assigns the value of XML-SCHEMA-PATH to SCHEMA-PATH before doing anything else. As a result, WebSpeed searches for your DTD.

How to use XML-SCHEMA-PATH

In WebSpeed, instead of accessing the SCHEMA-PATH attribute of X-document (of WEB-CONTEXT), access XML-SCHEMA-PATH. For example, if you want a WebSpeed application to set the DTD search path (perhaps based on a Web request) of an XML document, instead of having the application set the SCHEMA-PATH attribute of X-document (of WEB-CONTEXT), have the application set the XML-SCHEMA-PATH attribute (of WEB-CONTEXT).

For more information on accessing XML documents using the Document Object Model (DOM) and SAX interfaces, see *OpenEdge Development: Programming Interfaces*.

XML-SUPPRESS-NAMESPACE-PROCESSING attribute (WebSpeed Only)

Specifies whether to suppress namespace processing.

Data type: LOGICAL

Access: Readable/Writeable

Applies to: WEB-CONTEXT system handle

Almost identical to the SUPPRESS-NAMESPACE-PROCESSING attribute of the X-document handle. For more information on SUPPRESS-NAMESPACE-PROCESSING, see the reference entry for the SUPPRESS-NAMESPACE-PROCESSING attribute.

What XML-SUPPRESS-NAMESPACE-PROCESSING avoids

In WebSpeed, the first time you access the X-document handle or any of its attributes, either directly or indirectly, you trigger a load of the document, which precedes your access. For example, if you set the SUPPRESS-NAMESPACE-PROCESSING attribute of X-document (of WEB-CONTEXT) before accessing X-document (of WEB-CONTEXT) or any of its attributes, when the document arrives, WebSpeed first loads the document, then sets SUPPRESS-NAMESPACE-PROCESSING to your value. So at load time, SUPPRESS-NAMESPACE-PROCESSING might not be set to your value, which might cause document validation to fail.

By contrast, if you set XML-SUPPRESS-NAMESPACE-PROCESSING instead of SUPPRESS-NAMESPACE-PROCESSING, when the document arrives, WebSpeed assigns the value of XML-SUPPRESS-NAMESPACE-PROCESSING to SUPPRESS-NAMESPACE-PROCESSING before doing anything else. Even if this triggers a load of the document, SUPPRESS-NAMESPACE-PROCESSING already contains your value, so WebSpeed is able to validate your document.

How to use XML-SUPPRESS-NAMESPACE-PROCESSING

In WebSpeed, instead of accessing SUPPRESS-NAMESPACE-PROCESSING of X-document (of WEB-CONTEXT), access XML-SUPPRESS-NAMESPACE-PROCESSING (of WEB-CONTEXT). For example, if you want a WebSpeed application to turn namespace processing off (perhaps in response to a Web request), instead of having the application assign FALSE to SUPPRESS-NAMESPACE-PROCESSING of X-document (of WEB-CONTEXT) have the application assign FALSE to XML-SUPPRESS-NAMESPACE-PROCESSING (of WEB-CONTEXT).

For more information on accessing XML documents using the SAX and X-document interfaces, see *OpenEdge Development: Programming Interfaces*.

Y attribute

The pixel location of the top edge of the widget relative to the top edge of the parent widget or the display. The pixel location of the mouse cursor relative to the top edge of the display (for the last mouse event). This attribute is functionally equivalent to the ROW attribute.

Data type: INTEGER

Access: Readable/Writeable

Applies to: BROWSE widget (browse and cell), BUTTON widget, COMBO-BOX widget, CONTROL-FRAME widget, DIALOG-BOX widget, EDITOR widget, FIELD-GROUP widget, FILL-IN widget, FRAME widget, IMAGE widget, LAST-EVENT system handle, LITERAL widget, RADIO-SET widget, RECTANGLE widget, SELECTION-LIST widget, SLIDER widget, TEXT widget, TOGGLE-BOX widget, WINDOW widget

This attribute is read-only for field groups, browse cells, and the LAST-EVENT handle.

For all user interface widgets except windows, the Y attribute specifies the location, in pixels, of the top edge of the widget relative to the top edge of its parent widget. In windows, it is the location of the top edge of the window relative to the top edge of the display.

For a browse column, the Y attribute returns the Unknown value (?) if the column is hidden.

For control-frames, the Y attribute maps to the Top property of the control-frame COM object (ActiveX control container).

For the LAST-EVENT handle, the Y attribute returns the pixel location of a mouse event relative to the top edge of the current frame.

YEAR-OFFSET attribute

The current start date for the Progress two-digit year-range of 100 years. Use this attribute to display DATE, DATETIME, and DATETIME-TZ data when the format specifies a two-digit year.

Data type: INTEGER

Access: Readable/Writeable

Applies to: SESSION system handle

Typical values are 1920 or 1950. This attribute provides the same functionality as the Year Offset (-yy) parameter. The default value is 1950.

Events Reference

There are a number of factors that determine how Progress interprets events. The most important factor is the type of widget or handle receiving the event. Some widget types have default system actions in response to certain events. For example, the default system action for the A event on a fill-in widget is to insert the letter A into the fill-in at the current cursor location; however, there is no default system action for the A event on a button widget.

Different widget attribute settings determine how Progress interprets and prioritizes events. If you enable a widget for direct manipulation, direct manipulation events take priority over all other events. For example, if you write a trigger for a CHOOSE event and another for a SELECT event on a selectable widget, Progress only executes the SELECT event trigger when you click on that widget.

This chapter covers the following topics for user-interface events only:

- Introduction to Progress events
- Event tables

You may consider an event to be supported for all interfaces, on all operating systems, and for SpeedScript unless otherwise indicated in the reference entry. User-interface events do not apply to SpeedScript programming.

For information on the following events, see the relevant documentation:

- **DDE-NOTIFY** — *OpenEdge Development: Programming Interfaces*
- **PROCEDURE-COMPLETE** — *OpenEdge Application Server: Developing AppServer Applications*
- **WEB-NOTIFY** — *OpenEdge Application Server: Developing WebSpeed Applications*

Introduction to Progress events

This section covers the following topics:

- Event priority
- Applying events
- Triggers and low-level keyboard events

Event priority

The priority of events is an important concept. For any mouse or keyboard action on a widget, Progress generates a single event. Thus, certain events take priority over others that are generated by the same keyboard or mouse action for the same widget. Without direct manipulation, the priority (first to last) of keyboard events is key label, key function, and then high-level widget events such as CHOOSE. The priority of mouse events is three-button, portable, and then high-level widget events. Within three-button and portable mouse events, low-level mouse events (up, down) take priority over high-level mouse events (click, double-click). For more information on keyboard and mouse event priority, see the chapter on handling user input in *OpenEdge Development: Programming Interfaces*.

Applying events

You can apply any event to any widget using the APPLY statement. Depending on the event-widget pair, the APPLY statement may or may not perform the default system action. Regardless of whether there is a default system action associated with an event-widget pair, you can write a trigger for the pair. The APPLY statement executes a trigger associated with an event-widget pair. If the event-widget pair has a default system action, that action occurs before or after the trigger executes, depending on the event.

The APPLY statement also serves as a communications/dispatch mechanism between procedures in an application. You can define a trigger for an event-procedure pair.

```
ON CLOSE OF THIS-PROCEDURE
DO:
  APPLY "CLOSE" TO WINDOW-1.
END.
```

To define a trigger for a procedure, specify any Progress event in an ON statement for a procedure handle. This capability allows you to encapsulate functionality in a procedure. To access that functionality, simply use the APPLY statement to apply the appropriate event to the handle of the procedure. For more information, see the APPLY statement reference entry.

When working with browse widgets, you can apply events to the browse widget and to a browse cell in the currently focused row.

```
ON CHOOSE OF button1
DO:
  APPLY "ENTRY" TO my-browse IN FRAME a.
  /* Code to focus a particular row in the browse. */
  APPLY "ENTRY" TO column3 IN BROWSE my-browse.
END.
```

Since a browse cell is the intersection of a column and row, referencing the column name references the intersection of that column and the currently focused row.

Note: The most flexible technique for encapsulating functionality in a procedure is to define and call internal procedures of a persistent procedure. For more information, see the chapter on block properties in *OpenEdge Development: Progress 4GL Handbook*.

Triggers and low-level keyboard events

Some low-level keyboard events cannot have associated triggers and maintain their default behavior at the same time. In general, if Progress gets an event from the user interface system (UIS) that has a trigger associated with it, Progress handles the default behavior and tells the UIS to ignore the event. This allows Progress to cancel the default behavior in response to a RETURN NO-APPLY invoked by the trigger.

However, there are some low-level keyboard events for which Progress does not handle the default behavior. These include the cursor keys, especially. When Progress gets one of these events with an associated trigger, it tells the UIS to ignore the event as usual, but because Progress does not handle the default behavior for the event, the standard UIS behavior is lost, as well. Thus, a cursor key event (for example, CURSOR-UP) that has an associated trigger does not move the cursor.

Note that for many low-level events, such as mouse button and printable character events in fill-in fields and editors, Progress **does** provide the default handling. Triggers on these events have no effect on the default event behavior unless they return NO-APPLY. The same is true of keyboard events that generate high-level functions, such as TAB and RETURN.

For those low-level, non-printable, keyboard events that are not handled by Progress, do not associate triggers with them unless you do not want the default behavior of the event. For those low-level events that have no standard UIS behavior (such as, programmable function keys) triggers have no negative effects, and in fact, are very useful in defining a program action. In general, check any questionable low-level events in a test procedure both before and after associating triggers with them to see if any standard behavior is affected. An empty trigger block is sufficient to detect differences in behavior.

```
ON event ANYWHERE DO: END.
```

Event tables

The tables in this section describe user interface events, the user actions that generate the events, and widgets that have default behavior for the events. The term **field-level widgets** refers to any widgets that can be part of a field group in a frame: fill-ins, sliders, selection lists, toggle boxes, radio sets, editors, rectangles, images, text, buttons, combo boxes, and browse widgets. Frames, dialog boxes, windows, menus (including menu bars and pop-up menus), sub-menus, and menu items can also receive events. Note that there is frequently a distinction made between a browse widget and a single cell in an updateable browse. For the most part, a browse cell behaves as a fill-in widget.

The event tables in this section describe the following kinds of events:

- Keyboard events
- Mouse events
- High-level widget events
- Direct manipulation events
- Developer events
- Socket events
- ProDataSet events

Keyboard events

Progress makes all keyboard actions available as events that you can specify by either key label or key function. You can write triggers for these keyboard events, and associate these triggers with any field-level widget that receives input focus. For a complete list of key label and key function names, and information on how to use them, see the chapter on handling user input in *OpenEdge Development: Programming Interfaces*.

Keyboard events have default effects depending on the widget that receives the event. For example, the "A" key label event displays an uppercase "A" in a fill-in or editor widget, but has no default effect when applied to a button. Progress organizes some key function events into several classes that have default effects on selected groups of widgets. Progress also provides special keyboard events to write default triggers on classes of keys. You can use these default events to write a trigger for all keys in a particular class for which you have not defined a key label or key function event trigger.

Main classes of key function events

Progress supports three main classes of key function events:

- **Universal key function events** — These events apply to all user-interface widgets except menus, submenus, and menu items.
- **Navigation key function events** — These events apply to those field-level widgets that can receive focus.
- **Field editing key function events** — These events apply to fill-ins and browse cells.

Table 92 describes universal key function events.

Table 92: Universal key function events *(1 of 2)*

Event	Affected widgets	Progress action
BELL	All except control container, menu, menu item, and submenu.	Trigger dependent (typically used to execute the BELL statement).
END-ERROR	All except menu, menu item, and submenu.	For the first input operation of the program, raise the ENDKEY condition. For subsequent input operations, raise the ERROR condition.

Table 92: Universal key function events *(2 of 2)*

Event	Affected widgets	Progress action
ENDKEY	All except menu, menu item, and submenu.	Raise the ENDKEY condition.
ERROR	All except control container, menu, menu item, and submenu.	Raise the ERROR condition.
GO	All except menu, menu item, and submenu.	Submit the input values for this frame.
HELP	All except menu, menu item, and submenu.	Invoke application help.

Table 93 describes navigation key function events.

Table 93: Navigation key function events

Event	Affected widgets	Progress action
BACK-TAB	Browse, browse cell, button, combo box, control container, editor, fill-in, radio set, selection list, slider, toggle box.	Move focus to the previous widget in the tab order within the current frame family.
NEXT-FRAME	Browse, browse cell, button, combo box, editor, fill-in, radio set, selection list, slider, toggle box.	Move focus to the next frame parented by the active window.
PREV-FRAME	Browse, browse cell, button, combo box, editor, fill-in, radio set, selection list, slider, toggle box.	Move focus to previous frame parented by the active window.
TAB	Browse, browse cell, button, combo box, control container, editor, fill-in, radio set, selection list, slider, toggle box.	Move focus to the next widget in the tab order within the current frame family.

Table 94 describes field editing key function events.

Table 94: Field editing key function events

Event	Affected widgets	Progress action
BACKSPACE	Fill-in, browse cell.	Delete one character to the left.
CLEAR	Fill-in, browse cell.	Clear the current field value (character interfaces only).
DELETE-CHARACTER	Fill-in, browse cell.	Delete one character to the right.
RECALL	Fill-in, browse cell.	Restore the field to its value when it was last enabled.
RETURN	Fill-in, browse cell.	Default behavior is different for character and graphical interfaces and dependent on the DATA-ENTRY-RETURN attribute of the SESSION handle.

Default keyboard events

Progress provides two keyboard events that you can use to write default triggers. Table 95 describes these events.

Table 95: Default keyboard events

Event	Affected widgets	Meaning
ANY-KEY	Browse, browse cell, button, combo box, editor, fill-in, radio set, selection list, slider, toggle box.	Executes for any keyboard event for which the user has not defined a specific trigger.
ANY-PRINTABLE	Browse, browse cell, button, combo box, editor, fill-in, radio set, selection list, slider, toggle box.	Executes for any keyboard event that normally produces a printable character.

Mouse events

Progress supports two types of mouse events — portable and three-button events. You can use portable mouse events to associate triggers with logical actions of any mouse. You can use the three-button mouse events to associate triggers with specific physical actions of a three-button mouse.

The following tables reference portable mouse buttons for portable mouse events and physical mouse buttons for three-button mouse events. For more information on the mapping between portable and physical mouse buttons and how Progress processes mouse events in the 4GL, see the chapter on handling user input in *OpenEdge Development: Programming Interfaces*.

Portable mouse events

Table 96 lists the mouse events that apply to all mice, no matter how the buttons are configured.

Table 96: Portable mouse events *(1 of 3)*

Event	User action	Affected widgets	Progress action
MOUSE-SELECT-DOWN	Press the mouse SELECT button.	All	Trigger dependent.
MOUSE-SELECT-UP	Release the pressed mouse SELECT button.	All	Trigger dependent.
MOUSE-SELECT-CLICK	Press and release the mouse SELECT button.	All	Trigger dependent.
MOUSE-SELECT-DBLCLICK	Press and release the mouse SELECT button twice.	All	Trigger dependent.
MOUSE-MENU-DOWN	Press the mouse MENU button.	All	Trigger dependent.
MOUSE-MENU-UP	Release the pressed mouse MENU button.	All	Trigger dependent.
MOUSE-MENU-CLICK	Press and release the mouse MENU button.	All	Trigger dependent.
MOUSE-MENU-DBLCLICK	Press and release the mouse MENU button twice.	All	Trigger dependent.

Table 96: Portable mouse events *(2 of 3)*

Event	User action	Affected widgets	Progress action
MOUSE-EXTEND-DOWN	Press the mouse EXTEND button.	All	Trigger dependent.
MOUSE-EXTEND-UP	Release the pressed mouse EXTEND button.	All	Trigger dependent.
MOUSE-EXTEND-CLICK	Press and release the mouse EXTEND button.	All	Trigger dependent.
MOUSE-EXTEND-DBLCLICK	Press and release the mouse EXTEND button twice.	All	Trigger dependent.
MOUSE-MOVE-DOWN	Press the mouse MOVE button. **Note:** In Windows, a MOUSE-SELECT-DOWN trigger defined for the same widget takes priority over MOUSE-MOVE-DOWN.	All	Trigger dependent.
MOUSE-MOVE-UP	Release the pressed mouse MOVE button. **Note:** In Windows, a MOUSE-SELECT-UP trigger defined for the same widget takes priority over MOUSE-MOVE-UP.	All	Trigger dependent.

Table 96: Portable mouse events *(3 of 3)*

Event	User action	Affected widgets	Progress action
MOUSE-MOVE-CLICK	Press and release the mouse MOVE button. **Note:** In Windows, a MOUSE-SELECT-CLICK trigger defined for the same widget takes priority over MOUSE-MOVE-CLICK.	All	Trigger dependent.
MOUSE-MOVE-DBLCLICK	Press and release the mouse MOVE button twice. **Note:** In Windows, a MOUSE-SELECT-DBLCLICK trigger defined for the same widget takes priority over MOUSE-MOVE-DBLCLICK.	All	Trigger dependent.

Three-button mouse events

Table 97 lists the mouse events associated with physical mouse buttons.

Table 97: Three-button mouse events *(1 of 2)*

Event	User action	Affected widgets	Progress action
LEFT-MOUSE-DOWN	Press the left mouse button.	All	Trigger dependent.
LEFT-MOUSE-UP	Release the pressed left mouse button.	All	Trigger dependent.
LEFT-MOUSE-CLICK	Press and release the left mouse button.	All	Trigger dependent.
LEFT-MOUSE-DBLCLICK	Press and release the left mouse button twice.	All	Trigger dependent.

Table 97: Three-button mouse events *(2 of 2)*

Event	User action	Affected widgets	Progress action
RIGHT-MOUSE-DOWN	Press the right mouse button.	All	Trigger dependent.
RIGHT-MOUSE-UP	Release the pressed right mouse button.	All	Trigger dependent.
RIGHT-MOUSE-CLICK	Press and release the right mouse button.	All	Trigger dependent.
RIGHT-MOUSE-DBLCLICK	Press and release the right mouse button twice.	All	Trigger dependent.
MIDDLE-MOUSE-DOWN	Press the middle mouse button.	All	Trigger dependent.
MIDDLE-MOUSE-UP	Release the pressed middle mouse button.	All	Trigger dependent.
MIDDLE-MOUSE-CLICK	Press and release the middle mouse button.	All	Trigger dependent.
MIDDLE-MOUSE-DBLCLICK	Press and release the middle mouse button twice.	All	Trigger dependent.

High-level widget events

Table 98 lists high-level widget events. These are events generated by mouse or keyboard actions that perform high-level operations on a widget, such as entering a fill-in, choosing a button, or displaying a menu. Unless noted in the Progress Action column, triggers on these events execute before Progress applies the event. If the trigger returns NO-APPLY, Progress does not apply the event. If the trigger executes after the event takes place, NO-APPLY has no effect.

Note: If a CHOOSE, DEFAULT-ACTION, or VALUE-CHANGED event executes a trigger as a result of a mouse click that changes input focus, NO-APPLY will return focus to the widget that had focus prior to the event.

Table 98: High-level widget events *(1 of 6)*

Event	User action	Affected widgets	Progress action
CHOOSE	A keyboard or mouse action that chooses a widget.	Button, non-toggle-box menu item.	Trigger executes after choose takes place.
DEFAULT-ACTION	A native keyboard or mouse event that confirms the selection of a value in a selection list or browse. (In Windows applications, double-click a list item. In character applications, press **RETURN** or **DELETE-LINE**.)	Selection list, Browse.	Trigger dependent.
DROP-FILE-NOTIFY [2,3]	A mouse action that completes a drag-and-drop operation on a widget.	Browse, Button, Combo-box, Dialog-box, Editor, Fill-in, Frame, Radio-set, Selection-list, Slider, Toggle, Window	Trigger executes after drag-and-drop operation concludes. **Note:** The trigger should call the END-FILE-DROP () method when it has finished processing all the files.

Table 98: High-level widget events *(2 of 6)*

Event	User action	Affected widgets	Progress action
END-SEARCH [2]	Occurs when an updateable browse ends a user-initiated search when a user either selects a row marker or clicks in a cell.	Browse	Trigger dependent.
ENTRY	A keyboard or mouse action that gives focus to the widget.	Browse, browse cell, button, combo box, control container, dialog box, editor, fill-in, frame, radio set, selection list, slider, toggle box, window	Trigger dependent. **Note:** For a browse widget, ON ENTRY OF *browse-name* specifies a trigger for the browse widget and ON ENTRY OF *column-name* IN BROWSE *browse-name* specifies a trigger for a browse cell. The browse cell is the intersection of the named column and the currently focused row.
ITERATION-CHANGED	A keyboard or mouse action that changes the current iteration of a browse. This event is obsolete; see the VALUE-CHANGED Event reference entry.	Browse	Trigger dependent.

Table 98: High-level widget events *(3 of 6)*

Event	User action	Affected widgets	Progress action
LEAVE	A keyboard or mouse action that takes focus from the widget.	Browse, browse cell, button, combo box, control container, dialog box, editor, fill-in, frame, radio set, selection list, slider, toggle box, window	Trigger dependent. **Note:** For a browse widget, ON LEAVE OF *browse-name* specifies a trigger for the browse widget and ON LEAVE OF *column-name* IN BROWSE *browse-name* specifies a trigger for a browse cell. The browse cell is the intersection of the named column and the currently focused row.
MENU-DROP	A keyboard or mouse action that displays a menu.	Menu,[1] submenu	Trigger dependent.
OFF-END [4]	A keyboard or mouse action that tries to move after the last row of a browse.	Browse	Trigger dependent.
OFF-HOME	A keyboard or mouse action that tries to move before the first row of a browse.	Browse	Trigger dependent.
PARENT-WINDOW-CLOSE	An event that each descendant window receives when the common ancestor window in that family receives a WINDOW-CLOSE event.	Window	Trigger dependent.

Table 98: High-level widget events *(4 of 6)*

Event	User action	Affected widgets	Progress action
ROW-DISPLAY	Any browse action that results in a row being displayed in the browse.	Browse	Trigger dependent. **Note:** The use of triggers for this event is restricted to special cases. When a row is displayed, use a trigger to modify attributes of individual cells in the column. It should be restricted to the following uses: changing cell colors, changing the cell font, referencing the cell in an expression, and (in Windows) changing the cell format.
ROW-ENTRY	A keyboard or mouse action that gives an updateable cell focus in a browse row.	Browse	Trigger dependent.
ROW-LEAVE	A keyboard or mouse action that takes focus from the browse row where an updateable cell has focus.	Browse	Trigger dependent.
SCROLL-NOTIFY	A mouse action in the scrollbar area of a browse.	Browse	Trigger dependent. **Note:** This event allows the developer to track physical movement of the focused row in the browse viewport.

Table 98: High-level widget events *(5 of 6)*

Event	User action	Affected widgets	Progress action
START-SEARCH [2]	A keyboard or mouse action that places an updateable browse into search mode.	Browse	Trigger dependent.
VALUE-CHANGED	A keyboard or mouse action that changes the value of a widget. For the browse, any action that selects a row.	Browse, combo-box, editor (Windows GUI only), fill-in, radio set, selection list, slider, toggle box, toggle box menu item	Trigger executes after value changes.
WINDOW-CLOSE	A keyboard or mouse action that causes the native window manager to close the affected window or dialog box.	Dialog box, window	Trigger dependent.
WINDOW-MAXIMIZED [2]	A keyboard or mouse action that causes the native window system to resize the window to its maximum size.	Window	Trigger executes after event takes place. However since the native system has control, a NO-APPLY does not stop the event from occurring. **Note:** This event occurs only in Windows.

Table 98: High-level widget events *(6 of 6)*

Event	User action	Affected widgets	Progress action
WINDOW-MINIMIZED	A keyboard or mouse action that causes the native window system to minimize (iconify) a window and hide all of its descendant windows.	Window	Trigger executes after event takes place. However, since the native system has control, a NO-APPLY does not stop the event from occurring.
WINDOW-RESIZED	A keyboard or mouse action that causes the native window system to resize the window to any extent vertically or horizontally.	Window	Trigger executes after event takes place. However, since the native system has control, a NO-APPLY does not stop the event from occurring.
WINDOW-RESTORED	A keyboard or mouse action that causes the native window system to restore a window and any descendant windows to the state they were in before a prior maximize or minimize event.	Window	Trigger executes after event takes place. However since the native system has control, a NO-APPLY does not stop the event from occurring.

[1] Supported only of the Menu POPUP-ONLY attribute is set to TRUE and the menu is set as a popup for some other widget.

[2] Windows only.

[3] Graphical interfaces only.

[4] The OFF-END event can also occur when there are more rows to retrieve in the query on a ProDataSet temp-table buffer. For more information, see the "ProDataSet events" section on page 2195.

Direct manipulation events

Direct manipulation events are Progress events that directly modify the size, shape, position, and appearance of a widget. These events are generated by mouse actions. Each user interface widget either has direct manipulation enabled or does not. Some types of widgets, such as menus, cannot have direct manipulation enabled. You can enable widgets for direct manipulation by setting the SELECTABLE, MOVABLE, or RESIZABLE attribute to TRUE.

If a widget has direct manipulation enabled, then direct manipulation events take priority over all other events. In other words, while data manipulation is enabled, the widget cannot perform data entry or application control functions. For example, if you set SELECTABLE to TRUE for a button, Progress interprets a MOUSE-SELECT-UP event as a SELECTION event. If you set SELECTABLE to FALSE, Progress interprets the same event as a CHOOSE event.

Direct manipulation events can be broken down into two types: general and frame-only. General direct manipulation events apply to both field-level and frame widgets. Frame-only direct manipulation events apply only to frames.

The following sections list the Progress events associated with direct widget manipulation. The user actions listed for these events assume that you set the appropriate attributes to make each event possible. For example, a widget must be SELECTABLE to receive the SELECTION event. For more information on direct manipulation, see the chapter on direct manipulation in *OpenEdge Development: Progress 4GL Handbook*.

General direct manipulation events

Table 99 lists the direct manipulation events that apply to field-level widgets and frames:

Table 99: General direct manipulation events *(1 of 3)*

Event	User action	Affected widgets	Progress action
DESELECTION	For all selected widgets in a frame — Click the mouse SELECT button on an unselected widget or in empty space in the frame. For a single selected widget — Click the mouse EXTEND button on a selected widget.	Frame and field-level widgets with SELECTABLE attribute set to TRUE; browses.	**Internal:** Sets the widget's SELECTED attribute to FALSE. This setting takes effect after any trigger for the event executes. **Screen:** Removes the highlight from the affected widget or widgets.
END-MOVE	Release the pressed mouse MOVE button after moving the drag box for the widget or widgets.	Frame and field-level widgets with MOVABLE attribute set to TRUE; Also browse-columns.	**Internal:** Generates an END-MOVE event for each moved widget. **Screen:** Moves each widget to the new x and y coordinates of its drag box.
END-RESIZE	Release the pressed mouse SELECT button after stretching the resize box to resize the widget.	Frame and field-level widgets with RESIZABLE and SELECTABLE attributes set to TRUE; Also browse-columns.	**Internal:** Generates an END-RESIZE event for the resized widget. **Screen:** Resizes the widget to the new x and y coordinates of its resize box.

Table 99: General direct manipulation events *(2 of 3)*

Event	User action	Affected widgets	Progress action
END-ROW-RESIZE	Release the pressed mouse SELECT button after resizing a row.	Browses.	**Internal:** Generates an END-ROW-RESIZE event for the resized row. **Screen:** Resizes all rows to the new height.
SELECTION	For a single unselected widget—Click the mouse SELECT or EXTEND button on the widget. For multiple unselected widgets—Release the pressed EXTEND button after drawing a select box around the widgets.	Frame and field-level widgets with SELECTABLE attribute set to TRUE.	**Internal:** Sets each widget's SELECTED attribute to TRUE. This setting takes effect after any trigger for the event executes. **Screen:** Highlights the affected widget or widgets.
START-MOVE	For a single widget—With the mouse pointer on the widget, press and hold the mouse MOVE button, and begin moving the mouse pointer. For multiple selected widgets—With the mouse pointer on any one of the selected widgets, press and hold the mouse MOVE button, and begin moving the mouse pointer.	Frame and field-level widgets with MOVABLE attribute set to TRUE; for multiple widgets, SELECTABLE attribute also set to TRUE; Also browse-columns.	**Internal:** Sends a START-MOVE event to all selected widgets. If the trigger returns a NO-APPLY, Progress does not generate the subsequent END-MOVE event. **Screen:** Draws a drag box around each of the one or more affected widgets, and moves each drag box in the direction of the moving mouse pointer.

Table 99: General direct manipulation events *(3 of 3)*

Event	User action	Affected widgets	Progress action
START-RESIZE	With the mouse pointer on a resize handle of a selected widget, press and hold the mouse SELECT button and begin moving the mouse pointer.	Frame and field-level widgets with RESIZABLE and SELECTABLE attributes set to TRUE; Browse-columns .	**Internal:** Sends a START-RESIZE event to the selected widget. If the trigger returns NO-APPLY, Progress does not generate the subsequent END-RESIZE event. **Screen:** Stretches a resize box around the widget in the direction of the moving mouse pointer.
START-ROW-RESIZE	With the mouse pointer on a row, press and hold the mouse SELECT button and begin moving the mouse pointer.	Browses	**Internal:** Sends a START-ROW-RESIZE event to the browse. If the trigger returns NO-APPLY, Progress does not generate the subsequent END-ROW-RESIZE event. **Screen:** Stretches a resize box around the row in the direction of the moving mouse pointer.

Frame-only direct manipulation events

Table 100 lists the direct manipulation events that apply only to frames.

Table 100: Frame-only direct manipulation events *(1 of 2)*

Event	User action	Affected widgets	Progress action
EMPTY-SELECTION	Click the mouse SELECT button on an empty space in the frame.	Frame and dialog box, whether its SELECTABLE attribute is set to TRUE or FALSE.	**Internal:** Sends a DESELECTION event to all selected widgets in the frame and sends the EMPTY-SELECTION event to the frame. **Screen:** Removes the highlight around any selected widgets in the frame.

Table 100: Frame-only direct manipulation events *(2 of 2)*

Event	User action	Affected widgets	Progress action
END-BOX-SELECTION	Release the pressed mouse SELECT or EXTEND button after moving the mouse pointer to stretch the select box.	Frame and dialog box with BOX-SELECTABLE attribute set to TRUE.	**Internal:** If the user pressed the mouse SELECT button, Progress sends a SELECTION event to all widgets surrounded by the select box. If the user pressed a mouse EXTEND button, Progress sends a SELECTION event to all unselected widgets, and a DESELECTION event to all selected widgets surrounded by the select box. If a trigger on END-BOX-SELECTION returns NO-APPLY, Progress does not send a subsequent SELECTION or DESELECTION event. Note that this behavior differs from the behavior of END-MOVE and END-RESIZE. **Screen:** Erases the select box, highlights selected widgets, and removes the highlight from deselected widgets.
START-BOX-SELECTION	Press and hold the mouse SELECT or EXTEND button in an empty area of the frame and begin moving the mouse pointer.	Frame and dialog box with BOX-SELECTABLE attribute set to TRUE.	**Internal:** Sends a START-BOX-SELECTION event to the frame. If a trigger returns NO-APPLY, Progress does not generate the subsequent END-BOX-SELECTION event. **Screen:** Draws a select box, which initially appears as a dot.

Developer events

Progress provides ten events, labeled U1 through U10, that you can invoke on any widget using the APPLY statement. The only function of a developer event is the one provided by your own trigger definition.

Socket events

Progress looks for events to execute in the context of U/I blocking statements. During this processing if Progress detects that data is available on a socket or that the remote end closed its socket or it detects that a client has connected to a port that the server has enabled connections to, a socket event is generated.

There are only two socket events, READ-RESPONSE Event which applies only to socket objects and CONNECT Event which applies only to server socket objects.

READ-RESPONSE event

Progress Detects — Data is available on a socket or the remote end of a connection has closed its socket. Applies only to socket objects.

Progress Action — Progress invokes the READ-RESPONSE event procedure.

The SET-READ-RESPONSE-PROCEDURE( ) method is used to name the READ-RESPONSE event procedure and to associate it with a socket object. Progress invokes this procedure whenever it detects that data is available on the socket or that the remote end of the socket has closed its end of the socket. In this procedure, the SELF handle identifies the affected socket object.

To determine if the event procedure was invoked because data is available for reading or because of a disconnect, the application can use one of several methods:

- The CONNECTED( ) method returns FALSE if the socket is not connected to a port, TRUE if it is connected.
- The GET-BYTES-AVAILABLE( ) method returns zero if the socket is not connected to a port or the number of bytes available for reading if it is connected.
- The READ( ) method returns FALSE if the socket is not connected to a port. It returns TRUE and the read data if it is connected.

CONNECT event

Progress Detects — A client has connected to a port that the server has enabled connections to. Applies only to server socket objects.

Progress Action — Progress invokes the CONNECT event procedure.

The SET-CONNECT-PROCEDURE( ) method is used to name the CONNECT event procedure and to associate it with a server socket object. The CONNECT event procedure must accept one input parameter of type HANDLE. This is the handle to the implicitly created socket object for this connection. It is via this socket object that the server communicates with the client.

If the SET-CONNECT-PROCEDURE( ) method is not invoked, or if it fails, no connection procedure will be executed when the CONNECT event occurs.

ProDataSet events

Progress provides events you can invoke to execute application-specific code that handles FILL operations on a ProDataSet object or Buffer object, as well as row-level change operations. You can use the SET-CALLBACK-PROCEDURE( ) method to associate an action with these events.

Event procedures must define a single parameter for the ProDataSet object (DATASET or DATASET-HANDLE) as an INPUT parameter BY-REFERENCE. This allows the event procedure to operate on the ProDataSet object using static 4GL to reference its buffers and fields, without the ProDataSet object being physically copied. This also means that because the ProDataSet object is not copied, changes made to the ProDataSet object by the event procedure are made to the same copy used by all procedures.

The following sections describe the ProDataSet events:

- FILL events
- Row-level events
- OFF-END event
- FIND-FAILED event
- SYNCHRONIZE event

FILL events

There are two levels of FILL events: the first level is for a ProDataSet object or one of its member buffer objects; the second level is for individual records created in each temp-table.

Table 101 lists the first-level FILL events.

Table 101: First-level FILL events

Event	Affected objects	Progress action
AFTER-FILL	Buffer object of a DATASET temp-table, ProDataSet object.	This event occurs at the very end of a FILL, and can be used to adjust the contents of the ProDataSet object or Buffer object, reject the FILL operation, or disconnect from a server or database. For a child table, the event occurs once for each parent record that is created.
BEFORE-FILL	Buffer object of a DATASET temp-table, ProDataSet object.	This event occurs at the very beginning of a FILL, before anything is read or created. For a Buffer object, this event allows the developer to do preparatory work for an individual table. For the parent table in a set of related tables, where the FILL event is applied to this top-level table, it could be the same kind of connection code as for the ProDataSet object as a whole. For a child table, the event occurs once for each parent record that is created, and allows the developer to adjust the query for the child table, or cancel the FILL for that parent altogether. For a ProDataSet object, this event allows the developer to make a server or database connection, or do other preparatory work. Alternatively, it allows the developer to intercept and fully replace the default behavior.

Table 102 lists the second-level FILL events. These events occur once immediately before or after each record is created in a temp-table during a FILL.

Table 102: Second-level FILL events

Event	Affected objects	Progress action
AFTER-ROW-FILL	Buffer object of a DATASET temp-table.	This event occurs after a record is created in the temp-table. The procedure can, for example, modify field values in the record by supplying values for calculated fields, or perform filtering and reject a record by deleting it. The procedure cannot modify record currency using the ProDataSet object buffers in any other way. It can use separately defined buffers to modify the ProDataSet object in other ways. The procedure can RETURN ERROR to abort the entire FILL, or RETURN NO-APPLY to cancel the cascading of the FILL to child buffers, if any.
BEFORE-ROW-FILL	Buffer object of a DATASET temp-table.	This event occurs before a record is created in the temp-table, but after the data source record(s) for it are read. For example, this procedure could examine the database buffers or other information and decide not to create the record, using a RETURN NO-APPLY statement.

Row-level events

Row-level events are defined for making local changes to the records in a ProDataSet member buffer object.

Table 103 lists the row-level events.

Table 103: Row-level events *(1 of 2)*

Event	Affected objects	Progress action
ROW-CREATE	Buffer object of a DATASET temp-table.	This event occurs immediately after the record is created in the temp-table. The current buffer for the temp-table is available and contains initial values as defined in the temp-table definition (or inherited from the schema). You can use this event to calculate initial values for fields, make changes to other records, or reject the creation by deleting the new temp-table record.

Table 103: Row-level events *(2 of 2)*

Event	Affected objects	Progress action
ROW-DELETE	Buffer object of a DATASET temp-table.	This event occurs when you delete a temp-table record, immediately before the record is deleted. The event procedure can use this event to RETURN ERROR to cancel the delete, or to make adjustments to other records based on the delete. Since the record has not yet been deleted, the record is in the temp-table buffer and the code can look at its values. Because the code can assume that the DELETE will go through unless cancelled by the event procedure itself, it can take actions based on the record deletion while the record is still there to be looked at.
ROW-UPDATE	Buffer object of a DATASET temp-table.	This event occurs immediately before the record is updated in the temp-table. It typically occurs when: • The buffer scope ends. • The transaction scope ends. • The RELEASE statement or BUFFER-RELEASE( ) method is run on the buffer. • The buffer is needed for another record. Progress sets the SELF system handle to the handle of the buffer on which the event procedure is running before calling the event handler. If the event handler returns NO-APPLY or ERROR, the return is ignored. The handler has run, and it is too late to undo any changes to the record. You can use this event to determine if and how a record has changed by reading the buffer in the before-image table (using the SELF:BEFORE-ROWID attribute) and comparing it to the updated buffer. You can also use this event in the event handler to update fields in the record (for example, to supply a calculated field). You cannot read another record into the buffer on which the event procedure is running in the event handler. If you need to read another record, use a different buffer to avoid disturbing the record you are currently updating.

OFF-END event

The OFF-END event occurs when you position a query on a ProDataSet temp-table buffer past the last row. You can use this event to retrieve additional data source rows to add at the bottom of a ProDataSet temp-table (for example, in batches when there are too many data source rows to retrieve at one time).

The OFF-END event can also occur when the user performs a keyboard or mouse action in a browse that scrolls off the end (past the last row) of a browse on a ProDataSet temp-table buffer. For more information about using the OFF-END event with a browse, see the "High-level widget events" section on page 2182.

Note: The OFF-END event is similar to the QUERY-OFF-END attribute, which is set to TRUE whenever the associated query object is positioned past the last row. The difference is that you must test the QUERY-OFF-END attribute for this condition at a specific place in your application code, whereas the OFF-END event procedure executes like a trigger whenever the event occurs.

Consider the following restrictions when using the OFF-END event with a query on a ProDataSet temp-table buffer:

- You can attach these events only to a query on a single ProDataSet temp-table buffer. You cannot attach these events to a query on a database buffer, or a query that involves a join.
- The query must be a scrolling query.
- If you never RETURN NO-APPLY, from the OFF-END event handler, the query will infinitely loop.
- Call the SET-CALLBACK-PROCEDURE( ) method before the query is opened.
- If you use the GET LAST statement or GET-LAST( ) method to get the last record associated with the query, the event handler is called repeatedly until it **does not** RETURN NO-APPLY (indicating that all records have been retrieved). For this reason, use caution when offering users the GET LAST action.
- The INDEXED-REPOSITION option is ignored for the query.

FIND-FAILED event

The FIND-FAILED event occurs when a FIND on a ProDataSet temp-table buffer fails. This can be the result of the FIND statement (but not the FIND NEXT, FIND PREV, or FIND LAST statements, and not the CAN-FIND function), or the FIND-FIRST( ) or FIND-UNIQUE( ) methods (but not on the FIND-LAST( ) method).

You can use this event to adjust the contents of the ProDataSet object. The event handler must be able to determine the action to take based on the context of the ProDataSet object, and must RETURN NO-APPLY to indicate the action was successful. For example, when the event occurs, the event handler could retrieve a missing row or a set of related rows from the server automatically.

SYNCHRONIZE event

The SYNCHRONIZE event occurs when a ProDataSet temp-table buffer is synchronized. That is, when the SYNCHRONIZE( ) method is run on the buffer or a parent buffer, or the buffer is selected in a browse. The event handler is invoked recursively at every level of the ProDataSet object hierarchy just before the recursion to the child levels.

By default, if the query is associated with a browse, the synchronize action of reopening the query automatically refreshes the browse. If the query is not associated with a browse, the synchronize action automatically gets the first row in the query by invoking a GET FIRST operation. If there is a REPOSITION data relation and no browse, the synchronize action gets the next record in the query by invoking a GET NEXT operation. Once these actions attempt to populate the buffer at a particular level, the SYNCHRONIZE event runs before moving recursively to the next lower level.

This event allows you to fetch rows, display buffer values in a frame, or take some other action. The handler procedure can also RETURN NO-APPLY to cancel the cascading of the synchronization to child buffers.

Class Reference

This chapter contains reference descriptions for the following built-in Progress classes:

- Progress.Lang.Class class
- Progress.Lang.Object class

For more information about these built-in Progress classes, see *OpenEdge Getting Started: Object-oriented Programming*.

Progress.Lang.Class class

Progress.Lang.Class provides type information about a class or an interface. Progress provides a Progress.Lang.Class object instance for each user-defined class or interface in the OpenEdge session.

You can use the Progress.Lang.Class object instance to access class information from any object instance of a user-defined class using the following syntax:

```
proclass-reference = object-reference:GetClass( ).
```

Where *object-reference* is the object reference for an object instance of the class type for which to get type information, and *proclass-reference* is the object reference for the Progress.Lang.Class object instance. For example:

```
DEFINE VARIABLE myCustObj AS acme.myObjs.CustObj.
DEFINE VARIABLE myType AS CLASS Progress.Lang.Class.

myCustObj = NEW acme.myObjs.CustObj ( ).
myType = myCustObj:GetClass( ).
```

Progress.Lang.Class is FINAL and cannot be inherited by another class. You cannot delete a Progress.Lang.Class object instance.

Inherits Progress.Lang.Object class

Implements This class does not implement an interface.

Data members This class contains the following data members:

`Package AS CHARACTER`

The package portion of the user-defined class or interface type name. If the class or interface type name does not contain a package, the value of this data member is the Unknown value (?).

For more information about user-defined type names, see the Type-name syntax reference entry in this book.

This data member is read-only.

SuperClass AS Progress.Lang.Class

The object reference for the super class type information, if the user-defined class is a subclass. Otherwise, the value of this data member is the Unknown value (?).

This data member is read-only.

TypeName AS CHARACTER

The type name of the class or interface, which consists of the package and the class or interface name.

For more information about user-defined type names, see the Type-name syntax reference entry in this book.

This data member is read-only.

Methods

This class contains the following methods:

METHOD PUBLIC LOGICAL IsInterface ( )

If the class is defined as an interface, this method returns TRUE. Otherwise, it returns FALSE.

METHOD PUBLIC LOGICAL IsFinal ( )

If the class is defined as FINAL, this method returns TRUE. Otherwise, it returns FALSE.

Note

This class does not contain a constructor method. Thus, you cannot create an object instance of this class using the NEW statement. To obtain the object reference for the Progress.Lang.Class object instance associated with the current user-defined class object, you must use the GetClass( ) method in the Progress.Lang.Object class.

Progress.Lang.Object class

Progress.Lang.Object provides a common set of data members and methods that all classes inherit. This set of data members and methods let you write generic code to use with any user-defined class.

Progress.Lang.Object is the ultimate super class for all user-defined classes that do not explicitly inherit a super class using the INHERITS phrase.

You typically use this class to define a variable or parameter to represent more than one user-defined class type. For example:

- When defining a field in a temporary table as a class, you must specify the class as Progress.Lang.Object.
- When defining generic methods to use with object instances of different user-defined classes, you must define their parameters using the Progress.Lang.Object class.

Inherits This class does not inherit a super class.

Implements This class does not implement an interface.

Data members This class contains the following data members:

`NEXT-SIBLING AS Progress.Lang.Object`

The object reference for the next class object instance in the list of instances created in the current OpenEdge session. The value of this data member is available after obtaining a valid object reference (for example, by using the SESSION:FIRST-OBJECT attribute to obtain the object reference for the first class object instance in the list). If there are no class object instances in the current session, or you have gone past the last class object instance in the list, this attribute returns the Unknown value (?).

Once your position in the list is established, you can use the NEXT-SIBLING and PREV-SIBLING data members to walk the list of class object instances.

To check the validity of an object reference, use the VALID-OBJECT function.

This data member is read-only.

PREV-SIBLING AS Progress.Lang.Object

The object reference for the previous class object instance in the list of instances created in the current OpenEdge session. The value of this data member is available after obtaining a valid object reference (for example, by using the SESSION:LAST-OBJECT attribute to obtain the object reference for the last class object instance in the list). If there are no class object instances in the current session, or you have gone past the first class object instance in the list, this attribute returns the Unknown value (?).

Once your position in the list is established, you can use the NEXT-SIBLING and PREV-SIBLING data members to walk the list of class object instances.

To check the validity of an object reference, use the VALID-OBJECT function.

This data member is read-only.

Methods

This class contains the following methods:

CONSTRUCTOR PUBLIC Object ( )

Constructs an object instance of this class. Since this constructor method takes no parameters, Progress automatically invokes it when creating a subclass of this super class.

METHOD PUBLIC Progress.Lang.Object Clone ( )

Creates a copy of an object instance and returns an object reference for the copy.

This method has no default behavior. You must override this method in a user-defined class. If you invoke this method without overriding it, Progress generates an error message and returns the Unknown value (?).

METHOD PUBLIC LOGICAL Equals (*OtherObj* AS Progress.Lang.Object)

Compares the object reference for this class object instance to another object reference, where *OtherObj* is the other object reference with which to compare. If the object references are equivalent (that is, the data members of the two object instances are the same, or the two object references are referencing the same object instance), this method returns TRUE. Otherwise, it returns FALSE.

This method has no default behavior. You must override this method in a user-defined class. If you invoke this method without overriding it, Progress generates an error message and returns the Unknown value (?).

```
METHOD PUBLIC Progress.Lang.Class GetClass ( )
```

Returns the object reference for the Progress.Lang.Class object instance associated with the current user-defined class object instance.

```
METHOD PUBLIC CHARACTER ToString ( )
```

This method returns the class name of the object instance followed by a unique object identifier.

You typically override this method in a user-defined class to provide additional contextual information.

Note: This method is used by some Progress string functions and statements, such as the QUOTER and STRING functions and the MESSAGE and PUT statements.

Keyword Index

The following table lists all keywords and built-in object names in the Progress language. Built-in object names for procedure or database objects are listed in all lower case. The columns are as follows:

- **Keyword** — Specifies the full keyword or built-in object name.
- **Rsrv** — Indicates whether a keyword is reserved.
- **Minimum abbreviation** — Specifies the shortest abbreviation Progress recognizes for the keyword or name. If no abbreviation is specified, the keyword cannot be abbreviated.

Keyword	Rsrv	Minimum abbreviation
+	✓	–
-	✓	–
.	✓	–
&ELSE	✓	–
&ELSEIF	✓	–
&ENDIF	✓	–
&GLOBAL-DEFINE	✓	&GLOB
&IF	✓	–

Keyword	Rsrv	Minimum abbreviation
&MESSAGE	✓	–
&SCOPED-DEFINE	✓	&SCOP
&THEN	✓	–
&UNDEFINE	✓	&UNDEF
&WEBSTREAM	✓	–
*	✓	–
/	✓	–
:	✓	–
?	✓	–
@	✓	–
[	✓	–
]	✓	–
^	✓	–
{&BATCH-MODE}	–	{&BATCH}
{&FILE-NAME}	–	–
{&LINE-NUMBER}	–	{&LINE-NUMBE}
{&OPSYS}	–	–
{&SEQUENCE}	–	–
{&WINDOW-SYSTEM}	–	{&WINDOW-SYS}
’	✓	–
<	✓	–
<=	–	–

(3 of 69)

Keyword	Rsrv	Minimum abbreviation
<>	–	–
=	✓	–
>	✓	–
>=	–	–
ABSOLUTE	–	ABS
ACCELERATOR	–	–
ACCUM	✓	–
ACCUMULATE	✓	ACCUM
ACTIVE-WINDOW	✓	–
ADD	✓	–
ADD-BUFFER	–	–
ADD-CALC-COLUMN	–	–
ADD-COLUMNS-FROM	–	–
ADD-EVENTS-PROCEDURE	–	–
ADD-FIELDS-FROM	–	–
ADD-FIRST	–	–
ADD-INDEX-FIELD	–	–
ADD-LAST	–	–
ADD-LIKE-COLUMN	–	–
ADD-LIKE-FIELD	–	–
ADD-LIKE-INDEX	–	–
ADD-NEW-FIELD	–	–

Keyword	Rsrv	Minimum abbreviation
ADD-NEW-INDEX	–	–
ADD-SCHEMA-LOCATION	–	–
ADD-SUPER-PROCEDURE	–	–
ADM-DATA	–	–
ADVISE	–	–
ALERT-BOX	–	–
ALIAS	✓	–
ALL	✓	–
ALLOW-COLUMN-SEARCHING	–	–
ALLOW-REPLICATION	–	–
ALTER	✓	–
ALWAYS-ON-TOP	–	–
AMBIGUOUS	✓	AMBIG
ANALYZE	✓	ANALYZ
AND	✓	–
ANSI-ONLY	–	–
ANY	✓	–
ANYWHERE	–	–
APPEND	–	–
APPL-ALERT-BOXES	–	APPL-ALERT
APPL-CONTEXT-ID	–	–
APPLICATION	–	–

(5 of 69)

Keyword	Rsrv	Minimum abbreviation
APPLY	✓	–
APPSERVER-INFO	–	–
APPSERVER-PASSWORD	–	–
APPSERVER-USERID	–	–
ARRAY-MESSAGE	–	–
AS	✓	–
ASC	–	–
ASCENDING	✓	ASC
ASK-OVERWRITE	–	–
ASSIGN	✓	–
ASYNCHRONOUS	✓	–
ASYNC-REQUEST-COUNT	–	–
ASYNC-REQUEST-HANDLE	–	–
AT	✓	–
ATTACHED-PAIRLIST	–	–
ATTR-SPACE	✓	ATTR
AUDIT-CONTROL	✓	–
AUDIT-ENABLED	–	–
AUDIT-EVENT-CONTEXT	–	–
AUDIT-POLICY	✓	–
AUTHENTICATION-FAILED	–	–
AUTHORIZATION	✓	–

(6 of 69)

Keyword	Rsrv	Minimum abbreviation
AUTO-COMPLETION	–	AUTO-COMP
AUTO-ENDKEY	–	–
AUTO-END-KEY	–	–
AUTO-GO	–	–
AUTO-INDENT	–	AUTO-IND
AUTOMATIC	–	–
AUTO-RESIZE	–	–
AUTO-RETURN	✓	AUTO-RET
AUTO-SYNCHRONIZE	–	–
AUTO-ZAP	–	AUTO-Z
AVAILABLE	✓	AVAIL
AVAILABLE-FORMATS	–	–
AVERAGE	–	AVE
AVG	–	–
BACKGROUND	✓	BACK
BACKWARDS	–	BACKWARD
BASE64-DECODE	–	–
BASE64-ENCODE	–	–
BASE-ADE	–	–
BASE-KEY	–	–
BATCH-MODE	–	BATCH
BATCH-SIZE	–	–

(7 of 69)

Keyword	Rsrv	Minimum abbreviation
BEFORE-HIDE	✓	BEFORE-H
BEGIN-EVENT-GROUP	–	–
BEGINS	✓	–
BELL	✓	–
BETWEEN	✓	–
BGCOLOR	–	BGC
BIG-ENDIAN	✓	–
BINARY	–	–
BIND	–	–
BIND-WHERE	–	–
BLANK	✓	–
BLOCK-ITERATION-DISPLAY	–	–
BORDER-BOTTOM-CHARS	–	BORDER-B
BORDER-BOTTOM-PIXELS	–	BORDER-BOTTOM-P
BORDER-LEFT-CHARS	–	BORDER-L
BORDER-LEFT-PIXELS	–	BORDER-LEFT-P
BORDER-RIGHT-CHARS	–	BORDER-R
BORDER-RIGHT-PIXELS	–	BORDER-RIGHT-P
BORDER-TOP-CHARS	–	BORDER-T
BORDER-TOP-PIXELS	–	BORDER-TOP-P
BOX	–	–
BOX-SELECTABLE	–	BOX-SELECT

Keyword	Rsrv	Minimum abbreviation
BREAK	✓	–
BROWSE	–	–
BUFFER	–	–
BUFFER-CHARS	–	–
BUFFER-COMPARE	–	–
BUFFER-COPY	–	–
BUFFER-CREATE	–	–
BUFFER-DELETE	–	–
BUFFER-FIELD	–	–
BUFFER-HANDLE	–	–
BUFFER-LINES	–	–
BUFFER-NAME	–	–
BUFFER-RELEASE	–	–
BUFFER-VALUE	–	–
BUTTON	–	–
BUTTONS	–	BUTTON
BY	✓	–
BY-POINTER	–	–
BY-VARIANT-POINTER	–	–
CACHE	–	–
CACHE-SIZE	–	–
CALL	✓	–

(9 of 69)

Keyword	Rsrv	Minimum abbreviation
CALL-NAME	–	–
CALL-TYPE	–	–
CANCEL-BREAK	–	–
CANCEL-BUTTON	–	–
CAN-CREATE	–	–
CAN-DELETE	–	–
CAN-DO	✓	–
CAN-FIND	✓	–
CAN-QUERY	–	–
CAN-READ	–	–
CAN-SET	–	–
CAN-WRITE	–	–
CAPS	–	–
CAREFUL-PAINT	–	–
CASE	✓	–
CASE-SENSITIVE	✓	CASE-SEN
CAST	✓	–
CDECL	–	–
CENTERED	✓	CENTER
CHAINED	–	–
CHARACTER	–	CHAR
CHARACTER_LENGTH	–	–

Keyword	Rsrv	Minimum abbreviation
CHARSET	–	–
CHECK	✓	–
CHECKED	–	–
CHOOSE	–	–
CHR	✓	–
CLASS	–	–
CLASS-TYPE	–	–
CLEAR	✓	–
CLEAR-APPL-CONTEXT	–	–
CLEAR-LOG	–	–
CLEAR-SELECTION	–	CLEAR-SELECT
CLIENT-CONNECTION-ID	–	–
CLIENT-PRINCIPAL	–	–
CLIENT-TTY	–	–
CLIENT-TYPE	–	–
CLIENT-WORKSTATION	–	–
CLIPBOARD	✓	–
CLOSE	–	–
CLOSE-LOG	–	–
CODE	–	–
CODEBASE-LOCATOR	✓	–
CODEPAGE	–	–

(11 of 69)

Keyword	Rsrv	Minimum abbreviation
CODEPAGE-CONVERT	–	–
COLLATE	–	–
COL-OF	–	–
COLON	✓	–
COLON-ALIGNED	–	COLON-ALIGN
COLOR	✓	–
COLOR-TABLE	–	–
COLUMN	–	COL
COLUMN-BGCOLOR	–	–
COLUMN-DCOLOR	–	–
COLUMN-FGCOLOR	–	–
COLUMN-FONT	–	–
COLUMN-LABEL	✓	COLUMN-LAB
COLUMN-MOVABLE	–	–
COLUMN-OF	–	–
COLUMN-PFCOLOR	–	–
COLUMN-READ-ONLY	–	–
COLUMN-RESIZABLE	–	–
COLUMNS	✓	–
COLUMN-SCROLLING	–	–
COMBO-BOX	–	–
COM-HANDLE	–	–

Keyword	Rsrv	Minimum abbreviation
COMMAND	–	–
COMPARES	–	–
COMPILE	–	–
COMPILER	✓	–
COMPLETE	–	–
COM-SELF	–	–
CONFIG-NAME	–	–
CONNECT	–	–
CONNECTED	✓	–
CONSTRUCTOR	–	–
CONTAINS	–	–
CONTENTS	–	–
CONTEXT	–	–
CONTEXT-HELP	–	–
CONTEXT-HELP-FILE	–	–
CONTEXT-HELP-ID	–	–
CONTEXT-POPUP	–	–
CONTROL	✓	–
CONTROL-BOX	–	–
CONTROL-FRAME	–	–
CONVERT	–	–
CONVERT-3D-COLORS	–	–

(13 of 69)

Keyword	Rsrv	Minimum abbreviation
CONVERT-TO-OFFSET	–	CONVERT-TO-OFFS
COPY-DATASET	–	–
COPY-LOB	✓	–
COPY-TEMP-TABLE	–	–
COUNT	–	–
COUNT-OF	✓	–
CPCASE	–	–
CPCOLL	–	–
CPINTERNAL	–	–
CPLOG	–	–
CPPRINT	–	–
CPRCODEIN	–	–
CPRCODEOUT	–	–
CPSTREAM	–	–
CPTERM	–	–
CRC-VALUE	–	–
CREATE	✓	–
CREATE-LIKE	–	–
CREATE-NODE-NAMESPACE	–	–
CREATE-RESULT-LIST-ENTRY	–	–
CREATE-TEST-FILE	–	–
CURRENT	✓	–

Keyword	Rsrv	Minimum abbreviation
CURRENT_DATE	✓	–
CURRENT_DATE	–	–
CURRENT-CHANGED	✓	–
CURRENT-COLUMN	–	–
CURRENT-ENVIRONMENT	–	CURRENT-ENV
CURRENT-ITERATION	–	–
CURRENT-LANGUAGE	✓	CURRENT-LANG
CURRENT-RESULT-ROW	–	–
CURRENT-ROW-MODIFIED	–	–
CURRENT-VALUE	–	–
CURRENT-WINDOW	✓	–
CURSOR	✓	CURS
CURSOR-CHAR	–	–
CURSOR-LINE	–	–
CURSOR-OFFSET	–	–
DATABASE	✓	–
DATA-BIND	–	–
DATA-ENTRY-RETURN	–	DATA-ENTRY-RET
DATA-RELATION	✓	DATA-REL
DATASERVERS	✓	–
DATASET	✓	–
DATASET-HANDLE	✓	–

(15 of 69)

Keyword	Rsrv	Minimum abbreviation
DATA-SOURCE	✓	
DATA-SOURCE-COMPLETE-MAP	–	–
DATA-SOURCE-MODIFIED	–	–
DATA-TYPE	–	DATA-T
DATE	–	–
DATE-FORMAT	–	DATE-F
DAY	–	–
DBCODEPAGE	✓	–
DBCOLLATION	✓	–
DBNAME	✓	–
DBPARAM	✓	–
DB-REFERENCES	–	–
DBRESTRICTIONS	✓	DBREST
DBTASKID	✓	–
DBTYPE	✓	–
DBVERSION	✓	DBVERS
DCOLOR	–	–
DDE	✓	–
DDE-ERROR	–	–
DDE-ID	–	DDE-I
DDE-ITEM	–	–
DDE-NAME	–	–

Keyword	Rsrv	Minimum abbreviation
DDE-TOPIC	–	–
DEBLANK	✓	–
DEBUG	–	DEBU
DEBUG-ALERT	–	–
DEBUGGER	✓	–
DEBUG-LIST	✓	–
DECIMAL	–	DEC
DECIMALS	✓	–
DECLARE	✓	–
DECLARE-NAMESPACE	–	–
DECRYPT	–	–
DEFAULT	✓	–
DEFAULT-BUFFER-HANDLE	–	–
DEFAULT-BUTTON	–	DEFAUT-B
DEFAULT-COMMIT	–	–
DEFAULT-EXTENSION	–	DEFAULT-EX
DEFAULT-NOXLATE	✓	DEFAULT-NOXL
DEFAULT-WINDOW	✓	–
DEFINE	✓	DEF
DEFINED	–	–
DEFINE-USER-EVENT-MANAGER	–	–
DELETE	✓	DEL

(17 of 69)

Keyword	Rsrv	Minimum abbreviation
DELETE PROCEDURE	–	–
DELETE-CHARACTER	–	DELETE-CHAR
DELETE-CURRENT-ROW	–	–
DELETE-LINE	–	–
DELETE-RESULT-LIST-ENTRY	–	–
DELETE-SELECTED-ROW	–	–
DELETE-SELECTED-ROWS	–	–
DELIMITER	✓	–
DESC	–	–
DESCENDING	✓	DESC
DESELECT-FOCUSED-ROW	–	–
DESELECTION	–	–
DESELECT-ROWS	–	–
DESELECT-SELECTED-ROW	–	–
DESTRUCTOR	–	–
DIALOG-BOX	–	–
DICTIONARY	✓	DICT
DIR	–	–
DISABLE	✓	–
DISABLE-AUTO-ZAP	✓	–
DISABLED	–	–
DISABLE-DUMP-TRIGGERS	–	–

Keyword	Rsrv	Minimum abbreviation
DISABLE-LOAD-TRIGGERS	–	–
DISCONNECT	✓	DISCON
DISP	✓	–
DISPLAY	✓	DISP
DISPLAY-MESSAGE	–	–
DISPLAY-TYPE	–	DISPLAY-T
DISTINCT	✓	–
DO	✓	–
DOMAIN-DESCRIPTION	–	–
DOMAIN-NAME	–	–
DOMAIN-TYPE	–	–
DOS	✓	–
DOUBLE	–	–
DOWN	✓	–
DRAG-ENABLED	–	–
DROP	✓	–
DROP-DOWN	–	–
DROP-DOWN-LIST	–	–
DROP-FILE-NOTIFY	–	–
DROP-TARGET	–	–
DUMP	–	–
DYNAMIC	–	–

(19 of 69)

Keyword	Rsrv	Minimum abbreviation
DYNAMIC-FUNCTION	✓	–
EACH	✓	–
ECHO	–	–
EDGE-CHARS	–	EDGE
EDGE-PIXELS	–	EDGE-P
EDIT-CAN-PASTE	–	–
EDIT-CAN-UNDO	–	–
EDIT-CLEAR	–	–
EDIT-COPY	–	–
EDIT-CUT	–	–
EDITING	✓	–
EDITOR	–	–
EDIT-PASTE	–	–
EDIT-UNDO	–	–
ELSE	✓	–
EMPTY	–	–
EMPTY-TEMP-TABLE	–	–
ENABLE	✓	–
ENABLED-FIELDS	–	–
ENCODE	✓	–
ENCRYPT	–	–
ENCRYPT-AUDIT-MAC-KEY	–	–

Keyword	Rsrv	Minimum abbreviation
ENCRYPTION-SALT	–	–
END	✓	–
END-DOCUMENT	–	–
END-ELEMENT	–	–
END-EVENT-GROUP	–	–
END-FILE-DROP	–	–
ENDKEY	–	–
END-KEY	–	–
END-MOVE	–	–
END-RESIZE	–	–
END-ROW-RESIZE	–	–
END-USER-PROMPT	–	–
ENTERED	–	–
ENTRY	✓	–
EQ	–	–
ERROR	–	–
ERROR-COLUMN	–	ERROR-COL
ERROR-ROW	–	–
ERROR-STATUS	✓	ERROR-STAT
ESCAPE	✓	–
ETIME	✓	–
EVENT-GROUP-ID	–	–

(21 of 69)

Keyword	Rsrv	Minimum abbreviation
EVENT-PROCEDURE	✓	–
EVENT-PROCEDURE-CONTEXT	–	–
EVENTS	–	EVENT
EVENT-TYPE	–	EVENT-T
EXCEPT	✓	–
EXCLUSIVE-ID	–	–
EXCLUSIVE-LOCK	✓	EXCLUSIVE
EXCLUSIVE-WEB-USER	–	–
EXECUTE	–	–
EXISTS	✓	–
EXP	–	–
EXPAND	–	–
EXPANDABLE	–	–
EXPLICIT	–	–
EXPORT	✓	–
EXPORT-PRINCIPAL	–	–
EXTENDED	–	–
EXTENT	–	–
EXTERNAL	–	–
FALSE	✓	–
FETCH	✓	–
FETCH-SELECTED-ROW	–	–

Keyword	Rsrv	Minimum abbreviation
FGCOLOR	–	FGC
FIELD	✓	–
FIELDS	✓	FIELD
FILE	–	–
FILE-CREATE-DATE	–	–
FILE-CREATE-TIME	–	–
FILE-INFORMATION	✓	FILE-INFO
FILE-MOD-DATE	–	–
FILE-MOD-TIME	–	–
FILENAME	–	–
FILE-NAME	–	–
FILE-OFFSET	–	FILE-OFF
FILE-SIZE	–	–
FILE-TYPE	–	–
FILL	✓	–
FILLED	–	–
FILL-IN	–	–
FILTERS	–	–
FINAL	–	–
FIND	✓	–
FIND-BY-ROWID	–	–
FIND-CASE-SENSITIVE	✓	–

(23 of 69)

Keyword	Rsrv	Minimum abbreviation
FIND-CURRENT	–	–
FINDER	–	–
FIND-FIRST	–	–
FIND-GLOBAL	✓	–
FIND-LAST	–	–
FIND-NEXT-OCCURRENCE	✓	–
FIND-PREV-OCCURRENCE	✓	–
FIND-SELECT	✓	–
FIND-UNIQUE	–	–
FIND-WRAP-AROUND	✓	–
FIRST	✓	–
FIRST-ASYNCH-REQUEST	–	–
FIRST-CHILD	–	–
FIRST-COLUMN	–	–
FIRST-OBJECT	–	–
FIRST-OF	✓	–
FIRST-PROCEDURE	–	FIRST-PROC
FIRST-SERVER	–	–
FIRST-TAB-ITEM	–	FIRST-TAB-I
FIT-LAST-COLUMN	–	–
FIXED-ONLY	–	–
FLAT-BUTTON	–	–

Keyword	Rsrv	Minimum abbreviation
FLOAT	–	–
FOCUS	✓	–
FOCUSED-ROW	–	–
FOCUSED-ROW-SELECTED	–	–
FONT	✓	–
FONT-TABLE	–	–
FOR	✓	–
FORCE-FILE	–	–
FOREGROUND	–	FORE
FORM	✓	–
FORM INPUT	–	–
FORMAT	✓	FORM
FORMATTED	–	FORMATTE
FORM-LONG-INPUT	–	–
FORWARD	–	–
FORWARDS	–	FORWARD
FRAGMENT	–	FRAGMEN
FRAME	✓	FRAM
FRAME-COL	✓	–
FRAME-DB	✓	–
FRAME-DOWN	✓	–
FRAME-FIELD	✓	–

(25 of 69)

Keyword	Rsrv	Minimum abbreviation
FRAME-FILE	✓	–
FRAME-INDEX	✓	FRAME-INDE
FRAME-LINE	✓	–
FRAME-NAME	✓	–
FRAME-ROW	✓	–
FRAME-SPACING	–	FRAME-SPA
FRAME-VALUE	✓	FRAME-VAL
FRAME-X	–	–
FRAME-Y	–	–
FREQUENCY	–	–
FROM	✓	–
FROM-CHARS	✓	FROM-C
FROM-CURRENT	–	FROM-CUR
FROM-PIXELS	✓	FROM-P
FULL-HEIGHT-CHARS	–	FULL-HEIGHT
FULL-HEIGHT-PIXELS	–	FULL-HEIGHT-P
FULL-PATHNAME	–	FULL-PATHN
FULL-WIDTH-CHARS	–	FULL-WIDTH
FULL-WIDTH-PIXELS	–	FULL-WIDTH-P
FUNCTION	–	–
FUNCTION-CALL-TYPE	✓	–
GATEWAYS	✓	GATEWAY

Keyword	Rsrv	Minimum abbreviation
GE	–	–
GENERATE-MD5	–	–
GENERATE-PBE-KEY	–	–
GENERATE-PBE-SALT	–	–
GENERATE-RANDOM-KEY	–	–
GENERATE-UUID	–	–
GET	–	–
GET-ATTR-CALL-TYPE	✓	–
GET-ATTRIBUTE-NODE	–	–
GET-BINARY-DATA	–	–
GET-BLUE-VALUE	–	GET-BLUE
GET-BROWSE-COLUMN	–	–
GET-BUFFER-HANDLE	✓	–
GETBYTE	✓	–
GET-BYTE	✓	–
GET-CALLBACK-PROC-CONTEXT	–	–
GET-CALLBACK-PROC-NAME	–	–
GET-CGI-LIST	–	–
GET-CGI-LONG-VALUE	–	–
GET-CGI-VALUE	–	–
GET-CODEPAGES	✓	–
GET-COLLATIONS	✓	–

(27 of 69)

Keyword	Rsrv	Minimum abbreviation
GET-CONFIG-VALUE	–	–
GET-CURRENT	–	–
GET-DOUBLE	–	–
GET-DROPPED-FILE	–	–
GET-DYNAMIC	–	–
GET-FILE	–	–
GET-FIRST	–	–
GET-FLOAT	–	–
GET-GREEN-VALUE	–	GET-GREEN
GET-INDEX-BY-NAMESPACE-NAME	–	–
GET-INDEX-BY-QNAME	–	–
GET-ITERATION	–	–
GET-KEY-VALUE	✓	GET-KEY-VAL
GET-LAST	–	–
GET-LOCALNAME-BY-INDEX	–	–
GET-LONG	–	–
GET-MESSAGE	–	–
GET-NEXT	–	–
GET-NUMBER	–	–
GET-POINTER-VALUE	–	–
GET-PREV	–	–
GET-PRINTERS	–	–

Keyword	Rsrv	Minimum abbreviation
GET-PROPERTY	–	–
GET-QNAME-BY-INDEX	–	–
GET-RED-VALUE	–	GET-RED
GET-REPOSITIONED-ROW	–	–
GET-RGB-VALUE	–	–
GET-SELECTED-WIDGET	–	GET-SELECTED
GET-SHORT	–	–
GET-SIGNATURE	–	–
GET-SIZE	–	–
GET-STRING	–	–
GET-TAB-ITEM	–	–
GET-TEXT-HEIGHT-CHARS	–	GET-TEXT-HEIGHT
GET-TEXT-HEIGHT-PIXELS	–	GET-TEXT-HEIGHT-P
GET-TEXT-WIDTH-CHARS	–	GET-TEXT-WIDTH
GET-TEXT-WIDTH-PIXELS	–	GET-TEXT-WIDTH-P
GET-TYPE-BY-INDEX	–	–
GET-TYPE-BY-NAMESPACE-NAME	–	–
GET-TYPE-BY-QNAME	–	–
GET-UNSIGNED-SHORT	–	–
GET-URI-BY-INDEX	–	–
GET-VALUE-BY-INDEX	–	–
GET-VALUE-BY-NAMESPACE-NAME	–	–

(29 of 69)

Keyword	Rsrv	Minimum abbreviation
GET-VALUE-BY-QNAME	–	–
GET-WAIT-STATE	–	–
GLOBAL	✓	–
GO-ON	✓	–
GO-PENDING	✓	GO-PEND
GRANT	✓	–
GRAPHIC-EDGE	✓	GRAPHIC-E
GRID-FACTOR-HORIZONTAL	–	GRID-FACTOR-H
GRID-FACTOR-VERTICAL	–	GRID-FACTOR-V
GRID-SNAP	–	–
GRID-UNIT-HEIGHT-CHARS	–	GRID-UNIT-HEIGHT
GRID-UNIT-HEIGHT-PIXELS	–	GRID-UNIT-HEIGHT-P
GRID-UNIT-WIDTH-CHARS	–	GRID-UNIT-WIDTH
GRID-UNIT-WIDTH-PIXELS	–	GRID-UNIT-WIDTH-P
GRID-VISIBLE	–	–
GROUP	✓	–
GT	–	–
GUID	–	–
HANDLE	–	–
HANDLER	–	–
HAS-RECORDS	–	–
HAVING	✓	–

Keyword	Rsrv	Minimum abbreviation
HEADER	✓	–
HEIGHT-CHARS	–	HEIGHT
HEIGHT-PIXELS	–	HEIGHT-P
HELP	✓	–
HEX-DECODE	–	–
HEX-ENCODE	–	–
HIDDEN	–	–
HIDE	✓	–
HORIZONTAL	–	HORI
HOST-BYTE-ORDER	✓	–
HTML-CHARSET	–	–
HTML-END-OF-LINE	–	–
HTML-END-OF-PAGE	–	–
HTML-FRAME-BEGIN	–	–
HTML-FRAME-END	–	–
HTML-HEADER-BEGIN	–	–
HTML-HEADER-END	–	–
HTML-TITLE-BEGIN	–	–
HTML-TITLE-END	–	–
HWND	–	–
ICON	–	–
IF	✓	–

(31 of 69)

Keyword	Rsrv	Minimum abbreviation
IMAGE	–	–
IMAGE-DOWN	–	–
IMAGE-INSENSITIVE	–	–
IMAGE-SIZE	–	–
IMAGE-SIZE-CHARS	–	IMAGE-SIZE-C
IMAGE-SIZE-PIXELS	–	IMAGE-SIZE-P
IMAGE-UP	–	–
IMMEDIATE-DISPLAY	–	–
IMPLEMENTS	–	–
IMPORT	✓	–
IMPORT-PRINCIPAL	–	–
IN	✓	–
INCREMENT-EXCLUSIVE-ID	–	–
INDEX	✓	–
INDEXED-REPOSITION	–	–
INDEX-HINT	–	–
INDEX-INFORMATION	–	–
INDICATOR	✓	–
INFORMATION	–	INFO
IN-HANDLE	–	–
INHERITS	–	–
INITIAL	–	INIT

Keyword	Rsrv	Minimum abbreviation
INITIAL-DIR	–	–
INITIAL-FILTER	–	–
INITIALIZE-DOCUMENT-TYPE	–	–
INITIATE	–	–
INNER-CHARS	–	–
INNER-LINES	–	–
INPUT	✓	–
INPUT-OUTPUT	✓	INPUT-O
INPUT-VALUE	–	–
INSERT	✓	–
INSERT-ATTRIBUTE	–	–
INSERT-BACKTAB	–	INSERT-B
INSERT-FILE	–	–
INSERT-ROW	–	–
INSERT-STRING	–	–
INSERT-TAB	–	INSERT-T
INTEGER	–	INT
INTERFACE	–	–
INTERNAL-ENTRIES	–	–
INTO	✓	–
INVOKE	–	–
IS	✓	–

(33 of 69)

Keyword	Rsrv	Minimum abbreviation
IS-ATTR-SPACE	✓	IS-ATTR
IS-LEAD-BYTE	✓	IS-ATTR
IS-OPEN	–	–
IS-PARAMETER-SET	–	–
IS-ROW-SELECTED	–	–
IS-SELECTED	–	–
ITEM	–	–
ITEMS-PER-ROW	–	–
JOIN	✓	–
JOIN-BY-SQLDB	–	–
KBLABEL	✓	–
KEEP-CONNECTION-OPEN	–	–
KEEP-FRAME-Z-ORDER	–	KEEP-FRAME-Z
KEEP-MESSAGES	–	–
KEEP-SECURITY-CACHE	–	–
KEEP-TAB-ORDER	–	–
KEY	–	–
KEYCODE	✓	–
KEY-CODE	✓	–
KEYFUNCTION	✓	KEYFUNC
KEY-FUNCTION	✓	KEY-FUNC
KEYLABEL	✓	–

Keyword	Rsrv	Minimum abbreviation
KEY-LABEL	✓	–
KEYS	✓	–
KEYWORD	✓	–
KEYWORD-ALL	–	–
LABEL	✓	–
LABEL-BGCOLOR	–	LABEL-BGC
LABEL-DCOLOR	–	LABEL-DC
LABEL-FGCOLOR	–	LABEL-FGC
LABEL-FONT	–	–
LABEL-PFCOLOR	–	LABEL-PFC
LABELS	–	–
LANDSCAPE	–	–
LANGUAGES	–	LANGUAGE
LARGE	–	–
LARGE-TO-SMALL	–	–
LAST	✓	–
LAST-ASYNCH-REQUEST	–	–
LAST-BATCH	–	–
LAST-CHILD	–	–
LAST-EVENT	✓	LAST-EVEN
LASTKEY	✓	–
LAST-KEY	✓	–

(35 of 69)

Keyword	Rsrv	Minimum abbreviation
LAST-OBJECT	–	–
LAST-OF	✓	–
LAST-PROCEDURE	–	LAST-PROCE
LAST-SERVER	–	–
LAST-TAB-ITEM	–	LAST-TAB-I
LC	–	–
LDBNAME	✓	–
LE	–	–
LEAVE	✓	–
LEFT-ALIGNED	–	LEFT-ALIGN
LEFT-TRIM	–	–
LENGTH	–	–
LIBRARY	✓	–
LIKE	✓	–
LINE	–	–
LINE-COUNTER	✓	LINE-COUNT
LIST-EVENTS	–	–
LISTING	✓	LISTI
LIST-ITEM-PAIRS	–	–
LIST-ITEMS	–	–
LIST-PROPERTY-NAMES	–	–
LIST-QUERY-ATTRS	–	–

Keyword	Rsrv	Minimum abbreviation
LIST-SET-ATTRS	–	–
LIST-WIDGETS	–	–
LITERAL-QUESTION	–	–
LITTLE-ENDIAN	✓	–
LOAD	–	–
LOAD-DOMAINS	–	–
LOAD-ICON	–	–
LOAD-IMAGE	–	–
LOAD-IMAGE-DOWN	–	–
LOAD-IMAGE-INSENSITIVE	–	–
LOAD-IMAGE-UP	–	–
LOAD-MOUSE-POINTER	–	LOAD-MOUSE-P
LOAD-PICTURE	–	–
LOAD-SMALL-ICON	–	–
LOCAL-NAME	–	–
LOCATOR-COLUMN-NUMBER	–	–
LOCATOR-LINE-NUMBER	–	–
LOCATOR-PUBLIC-ID	–	–
LOCATOR-SYSTEM-ID	–	–
LOCATOR-TYPE	–	–
LOCKED	✓	–
LOCK-REGISTRATION	–	–

(37 of 69)

Keyword	Rsrv	Minimum abbreviation
LOG	–	–
LOG-AUDIT-EVENT	–	–
LOGICAL	–	–
LOGIN-EXPIRATION-TIMESTAMP	–	–
LOGIN-HOST	–	–
LOGIN-STATE	–	–
LOG-MANAGER	✓	–
LOGOUT	–	–
LOOKAHEAD	–	–
LOOKUP	✓	–
LT	–	–
MACHINE-CLASS	✓	–
MANDATORY	–	–
MANUAL-HIGHLIGHT	–	–
MAP	✓	–
MARGIN-EXTRA	–	–
MARGIN-HEIGHT-CHARS	–	MARGIN-HEIGHT
MARGIN-HEIGHT-PIXELS	–	MARGIN-HEIGHT-P
MARGIN-WIDTH-CHARS	–	MARGIN-WIDTH
MARGIN-WIDTH-PIXELS	–	MARGIN-WIDTH-P
MATCHES	–	–
MAX	–	–

Keyword	Rsrv	Minimum abbreviation
MAX-BUTTON	–	–
MAX-CHARS	–	–
MAX-DATA-GUESS	–	–
MAX-HEIGHT	–	–
MAX-HEIGHT-CHARS	–	MAX-HEIGHT-C
MAX-HEIGHT-PIXELS	–	MAX-HEIGHT-P
MAXIMIZE	–	–
MAXIMUM	–	MAX
MAX-ROWS	–	–
MAX-SIZE	–	–
MAX-VALUE	–	MAX-VAL
MAX-WIDTH	–	–
MAX-WIDTH-CHARS	–	MAX-WIDTH
MAX-WIDTH-PIXELS	–	MAX-WIDTH-P
MD5-DIGEST	–	–
MEMBER	✓	–
MEMPTR-TO-NODE-VALUE	–	–
MENU	–	–
MENUBAR	–	–
MENU-BAR	–	–
MENU-ITEM	–	–
MENU-KEY	–	MENU-K

(39 of 69)

Keyword	Rsrv	Minimum abbreviation
MENU-MOUSE	–	MENU-M
MERGE-BY-FIELD	–	–
MESSAGE	✓	–
MESSAGE-AREA	–	–
MESSAGE-AREA-FONT	–	–
MESSAGE-LINES	✓	–
METHOD	–	–
MIN	–	–
MIN-BUTTON	–	–
MIN-COLUMN-WIDTH-CHARS	–	MIN-COLUMN-WIDTH-C
MIN-COLUMN-WIDTH-PIXELS	–	MIN-COLUMN-WIDTH-P
MIN-HEIGHT-CHARS	–	MIN-HEIGHT
MIN-HEIGHT-PIXELS	–	MIN-HEIGHT-P
MINIMUM	–	MIN
MIN-SIZE	–	–
MIN-VALUE	–	MIN-VAL
MIN-WIDTH-CHARS	–	MIN-WIDTH
MIN-WIDTH-PIXELS	–	MIN-WIDTH-P
MODIFIED	–	–
MODULO	–	MOD
MONTH	–	–
MOUSE	✓	–

Keyword	Rsrv	Minimum abbreviation
MOUSE-POINTER	–	MOUSE-P
MOVABLE	–	–
MOVE-AFTER-TAB-ITEM	–	MOVE-AFTER
MOVE-BEFORE-TAB-ITEM	–	MOVE-BEFOR
MOVE-COLUMN	–	MOVE-COL
MOVE-TO-BOTTOM	–	MOVE-TO-B
MOVE-TO-EOF	–	–
MOVE-TO-TOP	–	MOVE-TO-T
MPE	✓	–
MULTI-COMPILE	–	–
MULTIPLE	–	–
MULTIPLE-KEY	–	–
MULTITASKING-INTERVAL	–	–
MUST-EXIST	–	–
NAME	–	–
NAMESPACE-PREFIX	–	–
NAMESPACE-URI	–	–
NATIVE	–	–
NE	–	–
NEEDS-APPSERVER-PROMPT	–	–
NEEDS-PROMPT	–	–
NEW	✓	–

(41 of 69)

Keyword	Rsrv	Minimum abbreviation
NEW-INSTANCE	–	–
NEW-ROW	–	–
NEXT	✓	–
NEXT-COLUMN	–	–
NEXT-PROMPT	✓	–
NEXT-ROWID	–	–
NEXT-SIBLING	–	–
NEXT-TAB-ITEM	–	NEXT-TAB-I
NEXT-VALUE	–	–
NO	✓	–
NO-APPLY	–	–
NO-ARRAY-MESSAGE	–	–
NO-ASSIGN	–	–
NO-ATTR-LIST	✓	NO-ATTR
NO-ATTR-SPACE	✓	NO-ATTR
NO-AUTO-VALIDATE	–	–
NO-BIND-WHERE	–	–
NO-BOX	–	–
NO-CONSOLE	–	–
NO-CONVERT	–	–
NO-CONVERT-3D-COLORS	–	–
NO-CURRENT-VALUE	–	–

Keyword	Rsrv	Minimum abbreviation
NO-DEBUG	–	–
NODE-VALUE-TO-MEMPTR	–	–
NO-DRAG	–	–
NO-ECHO	–	–
NO-EMPTY-SPACE	–	–
NO-ERROR	✓	–
NO-FILL	✓	NO-F
NO-FOCUS	✓	–
NO-HELP	–	–
NO-HIDE	✓	–
NO-INDEX-HINT	–	–
NO-JOIN-BY-SQLDB	–	–
NO-LABELS	✓	NO-LABEL
NO-LOBS	✓	–
NO-LOCK	✓	–
NO-LOOKAHEAD	–	–
NO-MAP	✓	–
NO-MESSAGE	✓	NO-MES
NONAMESPACE-SCHEMA-LOCATION	–	–
NONE	–	–
NO-PAUSE	✓	–
NO-PREFETCH	✓	NO-PREFE

(43 of 69)

Keyword	Rsrv	Minimum abbreviation
NORMALIZE	–	–
NO-ROW-MARKERS	–	–
NO-SCROLLBAR-VERTICAL	–	–
NO-SEPARATE-CONNECTION	–	–
NO-SEPARATORS	–	–
NOT	✓	–
NO-TAB-STOP	–	–
NO-UNDERLINE	–	NO-UND
NO-UNDO	✓	–
NO-VALIDATE	✓	NO-VAL
NOW	✓	–
NO-WAIT	✓	–
NO-WORD-WRAP	–	–
NULL	✓	–
NUM-ALIASES	✓	NUM-ALI
NUM-BUFFERS	–	–
NUM-BUTTONS	–	NUM-BUT
NUM-COLUMNS	–	NUM-COL
NUM-COPIES	–	–
NUM-DBS	✓	–
NUM-DROPPED-FILES	–	–
NUM-ENTRIES	✓	–

Keyword	Rsrv	Minimum abbreviation
NUMERIC	–	–
NUMERIC-FORMAT	–	NUMERIC-F
NUM-FIELDS	–	–
NUM-FORMATS	–	–
NUM-ITEMS	–	–
NUM-ITERATIONS	–	–
NUM-LINES	–	–
NUM-LOCKED-COLUMNS	–	NUM-LOCKED-COL
NUM-MESSAGES	–	–
NUM-PARAMETERS	–	–
NUM-REFERENCES	–	–
NUM-REPLACED	–	–
NUM-RESULTS	–	–
NUM-SELECTED-ROWS	–	–
NUM-SELECTED-WIDGETS	–	NUM-SELECTED
NUM-TABS	–	–
NUM-TO-RETAIN	–	–
NUM-VISIBLE-COLUMNS	–	–
OCTET-LENGTH	–	–
OF	✓	–
OFF	✓	–
OK	–	–

Keyword	Rsrv	Minimum abbreviation
OK-CANCEL	–	–
OLD	✓	–
ON	✓	
ON-FRAME-BORDER	–	ON-FRAME
OPEN	✓	–
OPSYS	✓	–
OPTION	✓	–
OR	✓	–
ORDERED-JOIN	–	–
ORDINAL	–	–
OS-APPEND	✓	–
OS-COMMAND	✓	–
OS-COPY	✓	–
OS-CREATE-DIR	✓	–
OS-DELETE	✓	–
OS-DIR	✓	–
OS-DRIVES	–	OS-DRIVE
OS-ERROR	–	–
OS-GETENV	–	–
OS-RENAME	✓	–
OTHERWISE	✓	–
OUTPUT	✓	–

Keyword	Rsrv	Minimum abbreviation
OVERLAY	✓	–
OVERRIDE	–	–
OWNER	–	–
PAGE	✓	–
PAGE-BOTTOM	✓	PAGE-BOT
PAGED	–	–
PAGE-NUMBER	✓	PAGE-NUM
PAGE-SIZE	–	–
PAGE-TOP	✓	–
PAGE-WIDTH	–	PAGE-WID
PARAMETER	✓	PARAM
PARENT	–	–
PARSE-STATUS	–	–
PARTIAL-KEY	–	–
PASCAL	–	–
PASSWORD-FIELD	✓	–
PATHNAME	–	–
PAUSE	✓	–
PBE-HASH-ALGORITHM	–	PBE-HASH-ALG
PBE-KEY-ROUNDS	–	–
PDBNAME	✓	–
PERSISTENT	✓	PERSIST

(47 of 69)

Keyword	Rsrv	Minimum abbreviation
PERSISTENT-CACHE-DISABLED	–	–
PFCOLOR	–	PFC
PIXELS	✓	–
PIXELS-PER-COLUMN	–	PIXELS-PER-COL
PIXELS-PER-ROW	–	–
POPUP-MENU	–	POPUP-M
POPUP-ONLY	–	POPUP-O
PORTRAIT	–	–
POSITION	–	–
PRECISION	–	–
PREFER-DATASET	–	–
PREPARED	–	–
PREPARE-STRING	–	–
PREPROCESS	✓	PREPROC
PRESELECT	–	PRESEL
PREV	–	–
PREV-COLUMN	–	–
PREV-SIBLING	–	–
PREV-TAB-ITEM	–	PREV-TAB-I
PRIMARY	–	–
PRINTER	–	–
PRINTER-CONTROL-HANDLE	–	–

Keyword	Rsrv	Minimum abbreviation
PRINTER-HDC	–	–
PRINTER-NAME	–	–
PRINTER-PORT	–	–
PRINTER-SETUP	–	–
PRIVATE	–	–
PRIVATE-DATA	–	PRIVATE-D
PRIVILEGES	✓	–
PROCEDURE	–	PROCE
PROCEDURE-CALL-TYPE	✓	–
PROCESS	✓	–
PROC-HANDLE	✓	PROC-HA
PROC-STATUS	✓	PROC-ST
proc-text	–	–
proc-text-buffer	–	–
PROFILER	✓	–
PROGRAM-NAME	✓	–
PROGRESS	✓	–
PROGRESS-SOURCE	–	PROGRESS-S
PROMPT	–	–
PROMPT-FOR	✓	PROMPT-F
PROMSGS	✓	–
PROPATH	✓	–

(49 of 69)

Keyword	Rsrv	Minimum abbreviation
PROTECTED	–	–
PROVERSION	✓	PROVERS
PROXY	–	–
PROXY-PASSWORD	–	–
PROXY-USERID	–	–
PUBLIC	–	–
PUBLIC-ID	–	–
PUBLISH	–	–
PUBLISHED-EVENTS	–	–
PUT	✓	–
PUTBYTE	✓	–
PUT-BYTE	✓	–
PUT-DOUBLE	–	–
PUT-FLOAT	–	–
PUT-KEY-VALUE	✓	PUT-KEY-VAL
PUT-LONG	–	–
PUT-SHORT	–	–
PUT-STRING	–	–
QUERY	✓	–
QUERY-CLOSE	✓	–
QUERY-OFF-END	✓	–
QUERY-OPEN	–	–

Keyword	Rsrv	Minimum abbreviation
QUERY-PREPARE	–	–
QUERY-TUNING	✓	–
QUESTION	–	–
QUIT	✓	–
QUOTER	–	–
RADIO-BUTTONS	–	–
RADIO-SET	–	–
RANDOM	–	–
RAW	–	–
RAW-TRANSFER	–	–
RCODE-INFORMATION	✓	RCODE-INFO
READ-AVAILABLE	✓	–
READ-EXACT-NUM	✓	–
READ-FILE	–	–
READKEY	✓	–
READ-ONLY	–	–
READ-XML	–	–
READ-XMLSCHEMA	–	–
REAL	–	–
RECID	✓	–
RECORD-LENGTH	–	–
RECTANGLE	✓	RECT

(51 of 69)

Keyword	Rsrv	Minimum abbreviation
RECURSIVE	–	–
REFERENCE-ONLY	–	–
REFRESH	–	–
REFRESHABLE	–	–
REFRESH-AUDIT-POLICY	–	–
REGISTER-DOMAIN	–	–
RELEASE	✓	–
REMOTE	–	–
REMOVE-EVENTS-PROCEDURE	–	–
REMOVE-SUPER-PROCEDURE	–	–
REPEAT	✓	–
REPLACE	–	–
REPLACE-SELECTION-TEXT	–	–
REPOSITION	✓	–
REPOSITION-BACKWARD	✓	–
REPOSITION-FORWARD	✓	–
REPOSITION-MODE	–	–
REPOSITION-TO-ROW	✓	–
REPOSITION-TO-ROWID	✓	–
REQUEST	–	–
RESET	–	–
RESIZABLE	–	RESIZA

Keyword	Rsrv	Minimum abbreviation
RESIZE	–	–
RESTART-ROWID	–	–
RETAIN	✓	–
RETAIN-SHAPE	–	–
RETRY	✓	–
RETRY-CANCEL	–	–
RETURN	✓	–
RETURN-INSERTED	–	RETURN-INS
RETURNS	✓	–
RETURN-TO-START-DIR	–	RETURN-TO-START-DI
RETURN-VALUE	–	RETURN-VAL
RETURN-VALUE-DATA-TYPE	–	–
REVERSE-FROM	–	–
REVERT	✓	–
REVOKE	✓	–
RGB-VALUE	–	–
RIGHT-ALIGNED	–	RETURN-ALIGN
RIGHT-TRIM	–	–
R-INDEX	✓	–
ROLES	–	–
ROUND	–	–
ROW	–	–

(53 of 69)

Keyword	Rsrv	Minimum abbreviation
ROW-HEIGHT-CHARS	–	HEIGHT
ROW-HEIGHT-PIXELS	–	HEIGHT-P
ROWID	✓	–
ROW-MARKERS	–	–
ROW-OF	–	–
ROW-RESIZABLE	–	–
RULE	–	–
RUN	✓	–
RUN-PROCEDURE	–	–
SAVE	✓	–
SAVE CACHE	–	–
SAVE-AS	–	–
SAVE-FILE	–	–
SAX-COMPLETE	✓	SAX-COMPLE
SAX-PARSE	–	–
SAX-PARSE-FIRST	–	–
SAX-PARSE-NEXT	–	–
SAX-PARSER-ERROR	✓	–
SAX-RUNNING	✓	–
SAX-UNINITIALIZED	✓	–
SAX-WRITE-BEGIN	✓	–
SAX-WRITE-COMPLETE	✓	–

Keyword	Rsrv	Minimum abbreviation
SAX-WRITE-CONTENT	✓	–
SAX-WRITE-ELEMENT	✓	–
SAX-WRITE-ERROR	✓	–
SAX-WRITE-IDLE	✓	–
SAX-WRITER	–	–
SAX-WRITE-TAG	✓	–
SCHEMA	✓	–
SCHEMA-LOCATION	–	–
SCHEMA-MARSHAL	–	–
SCHEMA-PATH	–	–
SCREEN	✓	–
SCREEN-IO	✓	–
SCREEN-LINES	✓	–
SCREEN-VALUE	–	SCREEN-VAL
SCROLL	✓	–
SCROLLABLE	–	
SCROLLBAR-HORIZONTAL	–	SCROLLBAR-H
SCROLL-BARS	–	–
SCROLLBAR-VERTICAL	–	SCROLLBAR-V
SCROLL-DELTA	–	–
SCROLLED-ROW-POSITION	–	SCROLLED-ROW-POS
SCROLLING	–	–

(55 of 69)

Keyword	Rsrv	Minimum abbreviation
SCROLL-OFFSET	–	–
SCROLL-TO-CURRENT-ROW	–	–
SCROLL-TO-ITEM	–	SCROLL-TO-I
SCROLL-TO-SELECTED-ROW	–	
SDBNAME	✓	–
SEAL	–	–
SEAL-TIMESTAMP	–	–
SEARCH	✓	–
SEARCH-SELF	✓	–
SEARCH-TARGER	✓	–
SECTION	–	–
SECURITY-POLICY	✓	–
SEEK	✓	–
SELECT	✓	–
SELECTABLE	–	–
SELECT-ALL	–	–
SELECTED	–	–
SELECT-FOCUSED-ROW	–	–
SELECTION	–	–
SELECTION-END	–	–
SELECTION-LIST	–	–
SELECTION-START	–	–

Keyword	Rsrv	Minimum abbreviation
SELECTION-TEXT	–	–
SELECT-NEXT-ROW	–	–
SELECT-PREV-ROW	–	–
SELECT-ROW	–	–
SELF	✓	–
SEND	–	–
send-sql-statement	–	send-sql
SENSITIVE	–	–
SEPARATE-CONNECTION	–	–
SEPARATOR-FGCOLOR	–	–
SEPARATORS	–	–
SERVER	–	–
SERVER-CONNECTION-BOUND	–	–
SERVER-CONNECTION-BOUND-REQUEST	–	–
SERVER-CONNECTION-CONTEXT	–	–
SERVER-CONNECTION-ID	–	–
SERVER-OPERATING-MODE	–	–
SESSION	✓	–
SESSION-ID	–	–
SET	✓	–
SET-APPL-CONTEXT	–	–
SET-ATTR-CALL-TYPE	✓	–

Keyword	Rsrv	Minimum abbreviation
SET-ATTRIBUTE-NODE	–	–
SET-BLUE-VALUE	–	SET-BLUE
SET-BREAK	–	–
SET-BUFFERS	–	–
SET-CALLBACK	–	–
SET-CLIENT	–	–
SET-COMMIT	–	–
SET-CONTENTS	–	–
SET-CURRENT-VALUE	–	–
SET-DB-CLIENT	–	–
SET-DYNAMIC	–	–
SET-EVENT-MANAGER-OPTION	–	–
SET-GREEN-VALUE	–	SET-GREEN
SET-INPUT-SOURCE	–	–
SET-OPTION	–	–
SET-OUTPUT-DESTINATION	–	–
SET-PARAMETER	–	–
SET-POINTER-VALUE	–	–
SET-PROPERTY	–	–
SET-RED-VALUE	–	SET-RED
SET-REPOSITIONED-ROW	–	–
SET-RGB-VALUE	–	–

Keyword	Rsrv	Minimum abbreviation
SET-ROLLBACK	–	–
SET-SELECTION	–	–
SET-SIZE	–	–
SETUSERID	✓	SETUSER
SET-WAIT-STATE	–	–
SHA1-DIGEST	–	–
SHARED	✓	–
SHARE-LOCK	✓	SHARE
SHOW-IN-TASKBAR	–	–
SHOW-STATS	✓	SHOW-STAT
SIDE-LABEL-HANDLE	–	SIDE-LABEL-H
SIDE-LABELS	–	SIDE-LAB
SILENT	–	–
SIMPLE	–	–
SINGLE	–	–
SIZE	–	–
SIZE-CHARS	–	SIZE-C
SIZE-PIXELS	–	SIZE-P
SKIP	✓	–
SKIP-DELETED-RECORD	✓	–
SLIDER	–	–
SMALL-ICON	–	–

(59 of 69)

Keyword	Rsrv	Minimum abbreviation
SMALLINT	–	–
SMALL-TITLE	–	–
SOME	✓	–
SORT	–	–
SOURCE	–	–
SOURCE-PROCEDURE	–	–
SPACE	✓	–
SQL	–	–
SQRT	–	–
SSL-SERVER-NAME	–	–
STANDALONE	–	–
START	–	–
START-DOCUMENT	–	–
START-ELEMENT	–	–
START-MOVE	–	–
START-RESIZE	–	–
START-ROW-RESIZE	–	–
STATE-DETAIL	–	–
STATUS	✓	–
STATUS-AREA	–	
STATUS-AREA-FONT	–	–
STDCALL	–	–

Keyword	Rsrv	Minimum abbreviation
STOP	–	–
STOP-PARSING	–	–
STOPPED	–	STOPPE
STORED-PROCEDURE	–	STORED-PROC
STREAM	✓	–
STREAM-IO	✓	–
STRETCH-TO-FIT	–	–
STRICT	–	–
STRING	–	–
STRING-VALUE	–	–
STRING-XREF	–	–
SUB-AVERAGE	–	SUB-AVE
SUB-COUNT	–	–
SUB-MAXIMUM	–	SUM-MAX
SUB-MENU	–	SUB-
SUB-MINIMUM	–	SUB-MIN
SUBSCRIBE	–	–
SUBSTITUTE	–	SUBST
SUBSTRING	–	SUBSTR
SUB-TOTAL	–	–
SUBTYPE	–	–
SUM	–	–

(61 of 69)

Keyword	Rsrv	Minimum abbreviation
SUPER	–	–
SUPER-PROCEDURES	–	–
SUPPRESS-NAMESPACE-PROCESSING	–	–
SUPPRESS-WARNINGS	–	SUPPRESS-W
SYMMETRIC-ENCRYPTION-ALGORITHM	–	–
SYMMETRIC-ENCRYPTION-IV	–	–
SYMMETRIC-ENCRYPTION-KEY	–	–
SYMMETRIC-SUPPORT	–	–
SYSTEM-ALERT-BOXES	–	SYSTEM-ALERT
SYSTEM-DIALOG	✓	–
SYSTEM-HELP	–	–
SYSTEM-ID	–	–
TABLE	✓	–
TABLE-HANDLE	–	–
TABLE-NUMBER	✓	–
TAB-POSITION	–	–
TAB-STOP	–	–
TARGET	–	–
TARGET-PROCEDURE	–	–
TEMP-DIRECTORY	–	TEMP-DIR
TEMP-TABLE	–	–
TEMP-TABLE-PREPARE	–	–

Keyword	Rsrv	Minimum abbreviation
TERM	✓	–
TERMINAL	✓	TERM
TERMINATE	–	–
TEXT	✓	–
TEXT-CURSOR	✓	–
TEXT-SEG-GROW	–	–
TEXT-SELECTED	–	–
THEN	✓	–
THIS-OBJECT	✓	–
THIS-PROCEDURE	✓	–
THREE-D	–	–
THROUGH	–	–
THRU	–	–
TIC-MARKS	–	–
TIME	✓	–
TIME-SOURCE	–	–
TITLE	✓	–
TITLE-BGCOLOR	–	TITLE-BGC
TITLE-DCOLOR	–	TITLE-DC
TITLE-FGCOLOR	–	TITLE-FGC
TITLE-FONT	–	TITLE-FO
TO	✓	–

(63 of 69)

Keyword	Rsrv	Minimum abbreviation
TODAY	–	–
TOGGLE-BOX	–	–
TOOLTIP	–	–
TOOLTIPS	–	–
TOPIC	–	–
TOP-ONLY	✓	–
TO-ROWID	✓	–
TOTAL	–	–
TRAILING	–	–
TRANS	✓	–
TRANSACTION	✓	–
TRANSACTION-MODE	–	–
TRANS-INIT-PROCEDURE	–	–
TRANSPARENT	–	–
TRIGGER	✓	–
TRIGGERS	✓	–
TRIM	✓	–
TRUE	✓	–
TRUNCATE	–	TRUNC
TYPE	–	–
TYPE-OF	–	–
UNBUFFERED	–	UNBUFF

Keyword	Rsrv	Minimum abbreviation
UNDERLINE	✓	UNDERL
UNDO	✓	–
UNFORMATTED	✓	UNFORM
UNION	✓	–
UNIQUE	✓	–
UNIQUE-ID	–	–
UNIQUE-MATCH	–	–
UNIX	✓	–
UNLESS-HIDDEN	–	–
UNLOAD	–	–
UNSUBSCRIBE	–	–
UP	✓	–
UPDATE	✓	–
URL	–	–
URL-DECODE	–	–
URL-ENCODE	–	–
URL-PASSWORD	–	–
URL-USERID	–	–
USE	–	–
USE-DICT-EXPS	–	–
USE-FILENAME	–	–
USE-INDEX	✓	–

(65 of 69)

Keyword	Rsrv	Minimum abbreviation
USER	–	–
USE-REVVIDEO	–	–
USERID	✓	–
USER-ID	–	–
USE-TEXT	–	–
USE-UNDERLINE	–	–
USE-WIDGET-POOL	–	–
USING	✓	–
V6DISPLAY	–	–
V6FRAME	✓	–
VALIDATE	–	–
VALIDATE-EXPRESSION	–	–
VALIDATE-MESSAGE	–	–
VALIDATE-SEAL	–	–
VALIDATION-ENABLED	–	–
VALID-EVENT	–	–
VALID-HANDLE	–	–
VALID-OBJECT	–	–
VALUE	✓	–
VALUE-CHANGED	✓	–
VALUES	✓	–
VARIABLE	–	VAR

Keyword	Rsrv	Minimum abbreviation
VERBOSE	–	–
VERSION	–	–
VERTICAL	–	VERT
VIEW	✓	–
VIEW-AS	✓	–
VIEW-FIRST-COLUMN-ON-REOPEN	–	–
VIRTUAL-HEIGHT-CHARS	–	VIRTUAL-HEIGHT
VIRTUAL-HEIGHT-PIXELS	–	VIRTUAL-HEIGHT-P
VIRTUAL-WIDTH-CHARS	–	VIRTUAL-WIDTH
VIRTUAL-WIDTH-PIXELS	–	VIRTUAL-WIDTH-P
VISIBLE	–	–
VOID	–	–
WAIT	–	–
WAIT-FOR	✓	
WARNING	–	–
WEB-CONTEXT	–	–
WEEKDAY	–	–
WHEN	✓	–
WHERE	✓	–
WHILE	✓	–
WIDGET	–	–
WIDGET-ENTER	–	WIDGET-E

Keyword	Rsrv	Minimum abbreviation
WIDGET-HANDLE	–	WIDGET-H
WIDGET-ID	–	–
WIDGET-LEAVE	–	WIDGET-L
WIDGET-POOL	–	–
WIDTH	–	–
WIDTH-CHARS	–	WIDTH
WIDTH-PIXELS	–	WIDTH-P
WINDOW	✓	–
WINDOW-MAXIMIZED	✓	WINDOW-MAXIM
WINDOW-MINIMIZED	✓	WINDOW-MINIM
WINDOW-NAME	–	–
WINDOW-NORMAL	✓	–
WINDOW-STATE	–	WINDOW-STA
WINDOW-SYSTEM	–	–
WITH	✓	–
WORD-INDEX	–	–
WORD-WRAP	–	–
WORK-AREA-HEIGHT-PIXELS	–	–
WORK-AREA-WIDTH-PIXELS	–	–
WORK-AREA-X	–	–
WORK-AREA-Y	–	–
WORKFILE	✓	–

Keyword	Rsrv	Minimum abbreviation
WORK-TABLE	✓	WORK-TAB
WRITE	✓	–
WRITE-CDATA	–	–
WRITE-CHARACTERS	–	–
WRITE-COMMENT	–	–
WRITE-DATA-ELEMENT	–	–
WRITE-EMPTY-ELEMENT	–	–
WRITE-ENTITY-REF	–	–
WRITE-EXTERNAL-DTD	–	–
WRITE-FRAGMENT	–	–
WRITE-MESSAGE	–	–
WRITE-PROCESSING-INSTRUCTION	–	–
WRITE-STATUS	–	–
WRITE-XML	–	–
WRITE-XMLSCHEMA	–	–
X	–	–
XCODE	✓	–
XML-DATA-TYPE	–	–
XML-NODE-TYPE	–	–
XML-SCHEMA-PATH	–	–
XML-SUPPRESS-NAMESPACE-PROCESSING	–	–
X-OF	–	–

Keyword	Rsrv	Minimum abbreviation
XREF	✓	–
Y	–	–
YEAR	–	–
YEAR-OFFSET	–	–
YES	✓	–
YES-NO	–	–
YES-NO-CANCEL	–	–
Y-OF	–	–

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y