


COMP-5: Integrating OLE Automation with Microsoft Office *Exchange* 2006 A Primer

About

Paul Guggenheim & Associates

- ❏ Working in Progress since 1984 and training Progress programmers since 1986
- ❏ Designed six comprehensive Progress courses covering all levels of expertise
- ❏ **TailorPro** Partner
- ❏ **Tools4Progress** Partner
- ❏ **Jargon** Authorized Reseller
- ❏ Major consulting clients include Bank One, Textron Fastening Systems, American Academy of Orthopaedic Surgeons and Foxwoods Casino

What is OLE?

 OLE Automation, or Object Linking and Embedding, is an effective way for a Progress® application to communicate with other applications in the MS Windows environment.

Before OLE

- ❏ The legacy method of communicating with a word processor or spreadsheet was to dump data to an ASCII file and then load the ASCII file into the other application.
- ❏ This process involved several manual steps, making it difficult to automate and secure against end user errors.

OLE Automation

- ❏ OLE Automation is a client/server based form of communication.
- ❏ An OLE Automation Server is an application that can be controlled by other applications.
- ❏ An OLE Automation Client is an application that can issue server commands.


OLE Automation

❏ Examples of OLE Automation Servers are:

- Microsoft Word
- Microsoft Excel
- Microsoft Outlook
- Microsoft Powerpoint
- Microsoft Mappoint


❏ Progress Versions 8.2 and higher are OLE Automation Clients.

OLE Information

 To learn about how to use OLE Automation, check the following:

- Progress COM Object Viewer
- Record Macros in the OLE Automation Server and look at the Visual Basic Application (VBA) source code produced.

OLE Information

 To learn about how to use OLE Automation, check the following:

- Visual Basic for Applications Help
 - Use the Object Browser to determine the value of constants
- Internet
 - MS Office Developer Website
(<http://www.msdn.microsoft.com/office>)

OLE Automation Servers

- ❏ The OLE Automation Server application consists of components called *objects* or *collections*. They relate to each other hierarchically.
- ❏ Each object consists of properties and methods that are analogous to Progress widget attributes and methods.

Using OLE in Progress – 3 Steps

1. Create an instance
2. Refer to other objects using properties and methods
3. Release Objects from memory

Using OLE in Progress

1. Create an instance

- Example:

```
def var hwordapp as com-handle.  
create "word.application" hwordapp.
```

- In order for an instance to be created, the server application must be registered to the same machine as the running Progress client, and the server program files must be located either on the same machine or a shared network drive.

Using OLE in Progress

CREATE statement

```
CREATE expression1 COM-hdl-var  
[CONNECT [TO expression2]]  
[NO-ERROR]
```

- *expression1* is a character string that names a unique Automation object in the system registry.
- *expression2* is a character string that identifies a file name of a particular application type, like **.doc** for Word files or **.xls** for Excel files.

Using OLE in Progress

2. Refer to other objects using properties and methods

– Example:

```
def var hdoc as com-handle.  
hdoc =  
    hwordapp.documents.open( "worddoc.doc" ) .
```

- **Documents** is a collection object belonging to the Word application, using the method **Open**.
- This method opens an existing Word document called "worddoc" and stores the reference to it in a variable called hdoc.

Using OLE in Progress

3. Release objects from memory

- Example:

 - `release object hdoc.`

 - `release object hwordapp.`

- Limits - Usually hundreds of server objects may be controlled by a client.


Using OLE in Progress

RELEASE statement

RELEASE OBJECT *COM-hdl-var*
[NO-ERROR]

- Once a COM object is released, any references to it *could* result in an invalid handle error.
 - **Warning:** Since a new COM object could be given the same handle as one that was deleted, you should always set *COM-hdl-var* to ‘?’ after releasing the object *and* use the **VALID-HANDLE** function to see if it refers to anything before referencing it or use the **NO-ERROR** keyword.

A Simple Example

-  The following program, **simple.p**, shows you how to open up a Microsoft Word document from within Progress.
- Notice we are using a **REPEAT** block to open more than one file at a time.

A Simple Example

```
create "Word.Application" oword.
```

- ❏ The **CREATE** statement produces a separate instance of the Microsoft Word application. This means that each time through the loop each file being opened will be in a separate Word *application* window.

A Simple Example

`message "before visible" view-as
alert-box information.`

`oword:visible = yes.`

- ❏ After the "before visible" message is executed, the Word application window appears. This is because the **VISIBLE** property for the Word application is set to YES.

A Simple Example

`message "before open" view-as
alert-box message.`

`word:Documents:Open(openfile).`


- ❏ After the "before open" message is run, the selected file is opened into view in the application window. This is because the **Open** method was used on the documents object collection for the Word application.

A Simple Example

release object oword.

- ❏ At the end of the **REPEAT** block, the object is released by Progress. This means that the Progress procedure is finished accessing the OLE object.

A Little-Less-Simple Example


 The program **simple2.p** improves on the previous example by showing how to open up a Microsoft Word document in the same application window from within Progress.

A Little-Less-Simple Example


Move the

create “Word.document” oword


above the repeat block.

 By moving the above statement outside the repeat block, all documents opened will be logically grouped in the same Word window. The windows pull down menu will show each document.

Working with Collections

 The program **simple3.p** augments the previous example by showing how to read and access the previously opened documents in a Microsoft Word application from within Progress.

Working with Collections

 The following illustrates the similarities of collections I have studied in Microsoft Automation Servers:

1. The collection name is plural, i.e. **Adjustments**, **Windows**, **Bookmarks**, **ChartObjects**, etc.
2. Each individual object inside of a collection usually is singular, i.e. **Window**, **Bookmark**, **ChartObject**, etc. (**Adjustment** doesn't exist).

Working with Collections

❏ The following illustrates the similarities of collections I have studied in Microsoft Automation Servers:

3. There is a read-only **Count** property that returns an integer representing the number of items in a collection.
4. There is an **Item** method that expects an integer representing the unique sequence number for a particular item in a collection.

Working with Collections

```
do i = 1 to oword:windows:count :  
    ...  
end. /* do i */
```

- ❏ Knowing this, a simple **DO** loop may be created to access all items in a collection.
- ❏ In **simple3.p**, after exiting the **REPEAT** block, a do loop is performed from 1 to **oword:Windows:Count** (number of documents open).

Working with Collections

```
create twordwin.
```

```
assign twordwin.tch =  
    oword:windows:item(i)
```

- ❏ A **twordwin** record is created and the **Item** method is used to assign the (i)th command handle to the **tch** temp-table field in the **twordwin** record.

Working with Collections

```
twordwin.tcaption =  
twordwin.tch:caption.
```

- ❏ The **Caption** property is similar to the **TITLE** attribute in Progress for a window or frame.

Working with Collections

on value-changed of b1
`tch:activate()`.

- Every time a new row is changed (**VALUE-CHANGED**) in the browse, that window in Word is brought to the foreground using the **Activate** method.

Working with Collections

```
on default-action of b1 do:  
    twordwin.tch:close().  
    delete twordwin.  
    run openq1.  
end.
```

- ❏ Every time a new row is selected through a carriage return or a mouse double-click (**DEFAULT-ACTION**) in the browse, that window in Word is closed.

Put a Spell on That Item

- ❏ Arguably, MS Word's most versatile automation application is spell check.
- ❏ In this example, **itemspell.p**, we are checking the spelling for the catalog description in the **item** table.

Put a Spell on That Item


```
a = item.cat-description:  
screen-value.
```

```
run spellit.p (input-output a).
```

- ❏ When the Spell button is selected, the **spellit.p** program is called with the text being passed as an input-output parameter.

Put a Spell on That Item

word:Documents:Add() .

 Inside **spellit.p**, it is necessary to add a document to word to perform the spell check.

Put a Spell on That Item

```
word:Documents:Item(1):Range  
(0,0):InsertAfter(spelltext).
```

- Next, our submitted text is placed into the document with the **InsertAfter** method.

Put a Spell on That Item

 The 4GL statement,

```
oword:Documents:Item(1):Range  
    (0,0):InsertAfter(spelltext).
```

is really a shortcut for the following:

```
oword:Documents:Add( ).
```

```
odoc = oword:Documents:Item(1).
```

```
orange = odoc:range(0,0).
```

```
orange:InsertAfter(spelltext).
```

– **Odoc** and **orange** are COM handles.


Put a Spell on That Item

word.Options:

CheckGrammarWithSpelling = true.

word.Documents.Item(1):

CheckGrammar().

 **CheckGrammarWithSpelling** is set to TRUE in order to check for both grammar and spelling with the **CheckGrammar** method.

Put a Spell on That Item

`oword.Visible = false.`

 **VISIBLE** was set to FALSE since we do not need to see Microsoft Word just to check spelling.

Put a Spell on That Item

```
if trim(spelltext) =  
    trim(oword:Selection:Text)  
then message "No spelling or  
grammar errors found or changed"  
view-as alert-box information.
```

- ❏ If nothing changed during the spell check then a message is issued.
- ❏ **TRIM** is needed since whitespace may be added by the **CheckGrammar** method.

Put a Spell on That Item

```
Else assign spelltext =  
    trim(oword:Selection:Text).
```

- ❏ Otherwise we assign the selected text by first using the **Selection** method, then using the **Text** property for the selection object.


Put a Spell on That Item

`oword.Quit(0).`

`release object oword.`

- ❏ Finally we close the application and release the handle.

Mail Merge Made Easy

 Mail Merge is easy if you remember to use bookmarks.

Mail Merge Made Easy

❏ Let's examine the following example:

- **Cimerge.p** contains two browses based on the following temp-tables: **tcustomer** for the first browse and **titem** and **tcustitem** for the second one.
 - **Tcustitem** contains the products that a customer has ordered and their total monetary value.

Mail Merge Made Easy

- ❏ When a customer is selected in the left browse, all items for that customer are shown in the second browse listed in descending order from largest monetary value to smallest. Also the address, sales rep, and contact information are displayed in the frame.
- ❏ After an item for that customer is selected, a mail merge is performed.

Mail Merge Made Easy

```
odoc = oword.Documents:  
Open( "e:\ole\merge.doc" ).
```


- ❏ After the application is opened for the first time, the merge document is opened into Word.

Mail Merge Made Easy

```
orange = odoc:  
    GoTo(-1, , , "Customer").
```

- ❏ This allows us to use the **GoTo** method to place the cursor on the desired bookmark.
 - This method returns the position (range) where the bookmark is located within the document and allows us to use the **InsertAfter** method at that position.


Mail Merge Made Easy

-  The **GoTo** method requires four arguments. The first is the type of object we are going to. **-1** is for bookmarks.
- How did I know this?
 - Well, here is where the Visual Basic help is crucial.


Mail Merge Made Easy

- ❏ First select MS Word Visual Basic Help.
The easiest way to do this is to select Edit Macros from the Tools menu in word. From the Edit Macros menu, lookup the **GoTo** method. You will see several constants under *What*, the first **GoTo** argument.

Mail Merge Made Easy

 The constant desired in this case is **wdGoToBookmark**. In order to find its value, we need to select the *Object Browser* from the Visual Basic button bar. We then search for **wdGoToBookmark** and find a value of **-1**.

Mail Merge Made Easy

 The next two arguments are skipped because they are not needed. The last argument, **Name**, is the name of the bookmark.

Mail Merge Made Easy

orange:insertparagraphafter().

❏ To insert a carriage return we just use the **InsertParagraphAfter** method.

❏ This is used for:

1. Inserting the contact name and customer name at the same bookmark
2. Inserting a non-blank second address line below the first address line.

Mail Merge Made Easy


```
savfile = "e:\ole\tmpmerge.doc".  
odoc.SaveAs(savfile).  
odoc.Close().
```

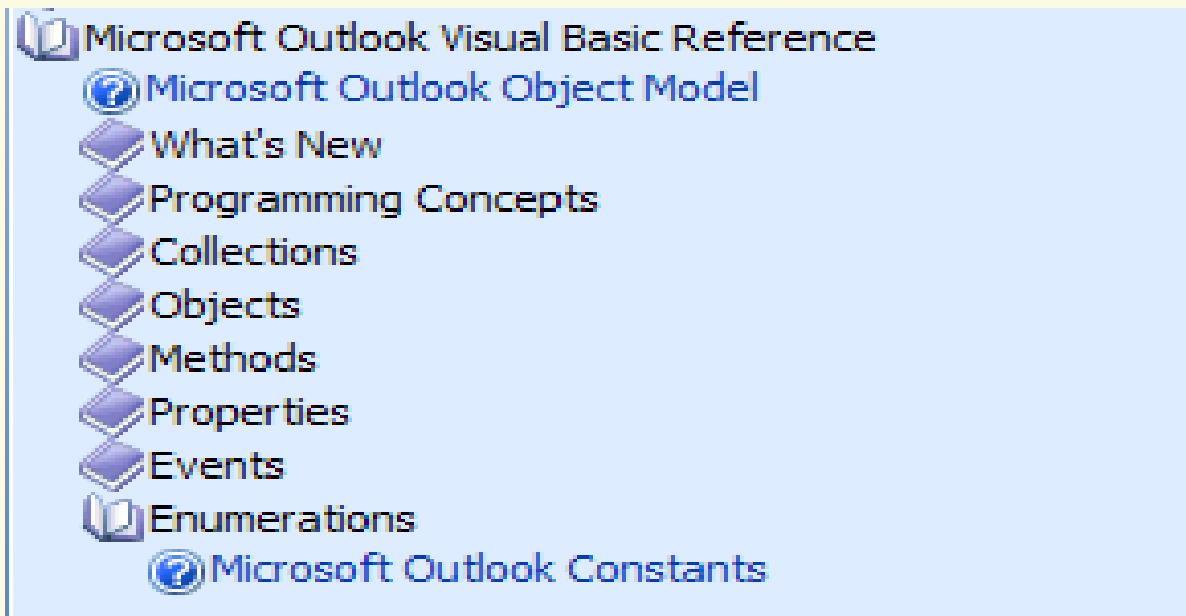
- ❏ The second time we do the merge, I save the merge document to a temporary one called **tmpmerge.doc** and close it. That way, I can reopen the merge document as many times as desired.

Outlook Synchronization

- ❏ Another handy application is to synchronize your customer list with Outlook's Contact Folder.
- ❏ Outlook provides a hierarchical object model in the application help.
- ❏ It lists the objects, properties, methods and events.

Outlook Synchronization

 From Outlook help, select Table of Contents, then Microsoft Outlook Visual Basic Reference.



Outlook Synchronization

❏ The last selection, Enumerations can replace the object browser for looking up constants.

❏ For example, it lists the valid item types:

▼ **OlItemType**

Constant	Value
olAppointmentItem	1
olContactItem	2
olDistributionListItem	7
olJournalItem	4
olMailItem	0
olNoteItem	5
olPostItem	6
olTaskItem	3

Outlook Synchronization

❏ The procedure, `olsync.p`, shows how to synchronize the sports customer table with the Outlook contact folder.

❏ To instantiate Outlook, it does the following:

```
create "Outlook.Application" ool.  
ons = ool:GetNameSpace("MAPI").  
octf = ons:getdefaultfolder(10).  
octf:display().
```

Outlook Synchronization

- ❏ Next, the user highlights the desired customers in the browse that are to be transferred or deleted to/from Outlook.
- ❏ When the add button is pressed, for each customer selected, the findcontact procedure is run to see if the customer has already been added. This is accomplished using the find method on the Outlook account field.

Outlook Synchronization

- ❏ Next the updatecontact procedure is run. If the find method fails then the octitem is not valid and the createitem method is executed and the customer number is stored in the account field.
- ❏ The rest of the outlook fields are updated and the contact is saved.

Outlook Synchronization

- ❏ For removing entries from the contact folder, the delete button is pressed.
- ❏ The delete button trigger runs findcontact for each customer selected and then executes the delete method for valid contacts found.

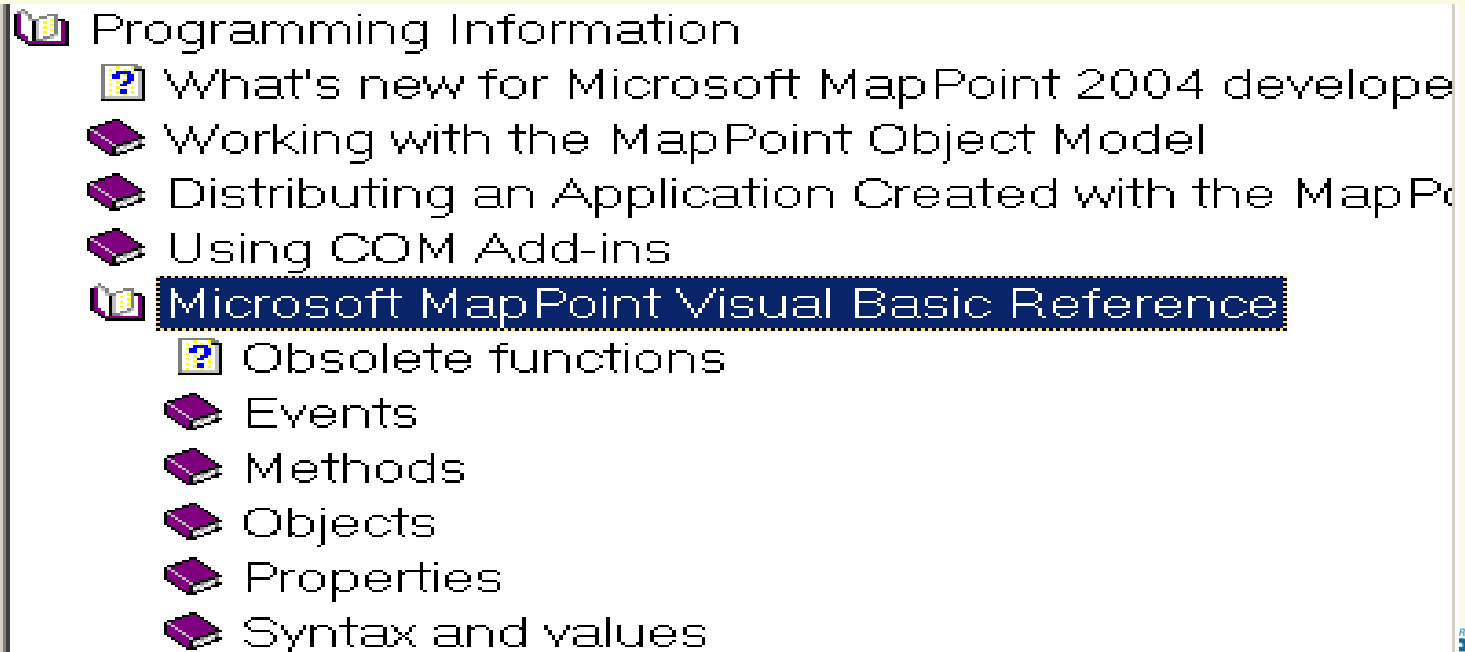
On the road again

- ❏ One of the more productive uses of OLE is mixing address data with a map program such as Microsoft Mappoint.
- ❏ Since Mappoint emerged a little later than the previously shown applications, it's Visual Basic documentation for OLE is more comprehensive and easier to follow.



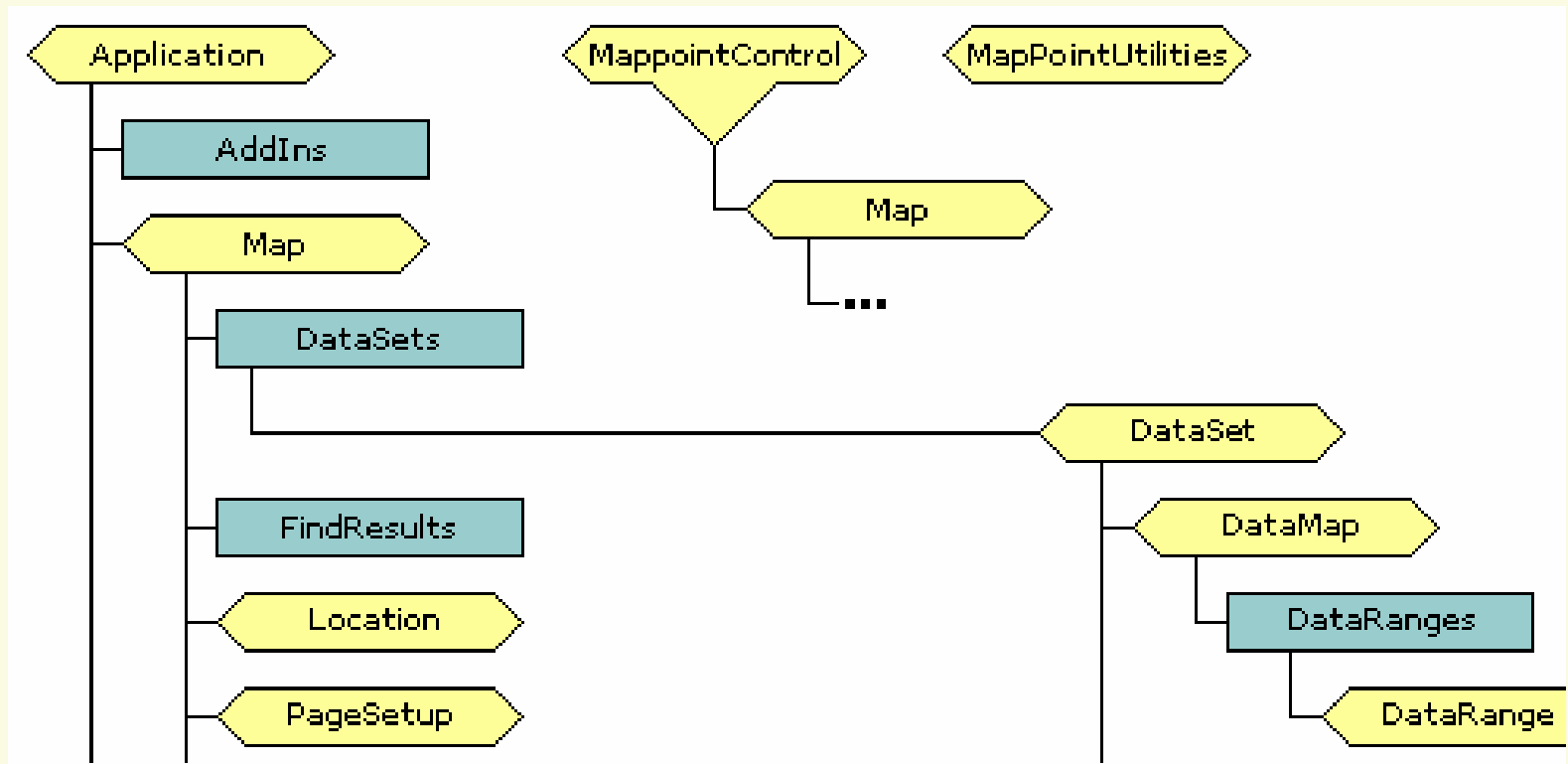
On the road again

- From Mappoint help, select contents, programming information, then Visual Basic Reference.



On the road again

 An object model map is displayed.



On the road again

- ❏ The procedure `mpl.p` shows some of the basic properties and methods to working with Mappoint.
- ❏ First, the application is invoked with:

Create "MapPoint.application" omp.

On the road again

- ❏ Next, the activemap handle is obtained.
- ❏ Then, findaddressresults method is used on the activemap handle which returns the handle to the find address result collection.

```
ofar = oam:findaddressresults  
("1788 Second Street", "Highland  
Park", "", "IL", "").
```

On the road again

- ❏ The postal-code is not necessary, but can also be used to improve the search.
- ❏ Since this collection could contain more than one result, only the first result is assigned. The resultsquality property will be covered in a later example.

```
oitem = ofar:item(1).
```


On the road again

- ❏ The Goto method is what zooms the map to the selected address.
- ❏ Next, a pushpin is added to the address.
- ❏ Finally, the balloon state is set so that the address information is displayed next to the pushpin.
- ❏ The quit method is used to close Mappoint.

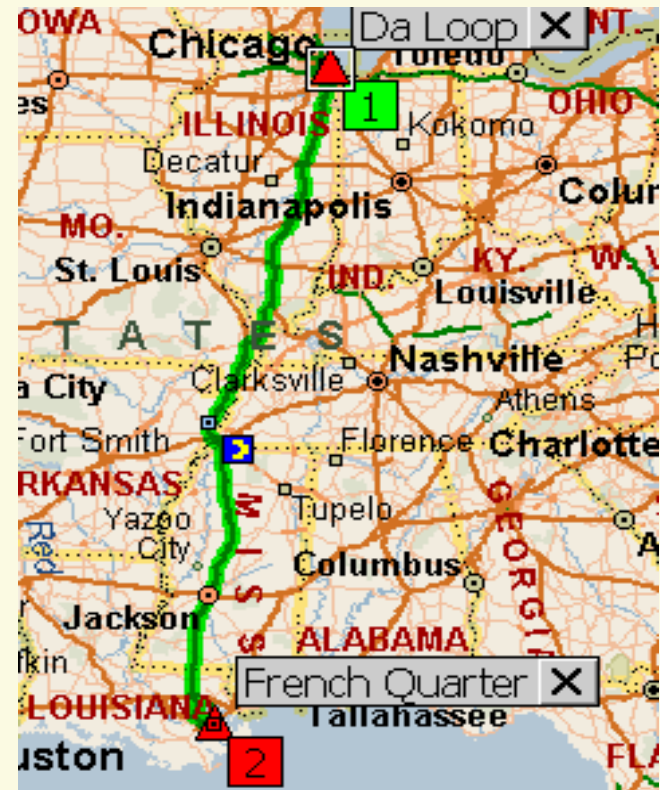
I've been everywhere

- ❏ In mp2.p, two addresses are entered and the distance and route are calculated.
- ❏ This example uses the ofar com-handle with an extent of 2 for the two addresses.
- ❏ The distance method calculates the distance by accepting both findaddressresults handles as parameters.
- ❏ Notice that units may be in miles or kilometers.



I've been everywhere

- ❏ The active route handle or t is obtained from the activemap handle.
- ❏ For the route, two waypoints are added and then the calculate method is used.



Return to sender

- ❏ In mpbr.p, the user can enter in multiple addresses. Each address is validated and then displayed in a multiple selection browse.
- ❏ Two addresses may be selected to calculate distance and routing.
- ❏ The application coordinates are set to fit both the progress and mappoint applications side by side.

Return to sender

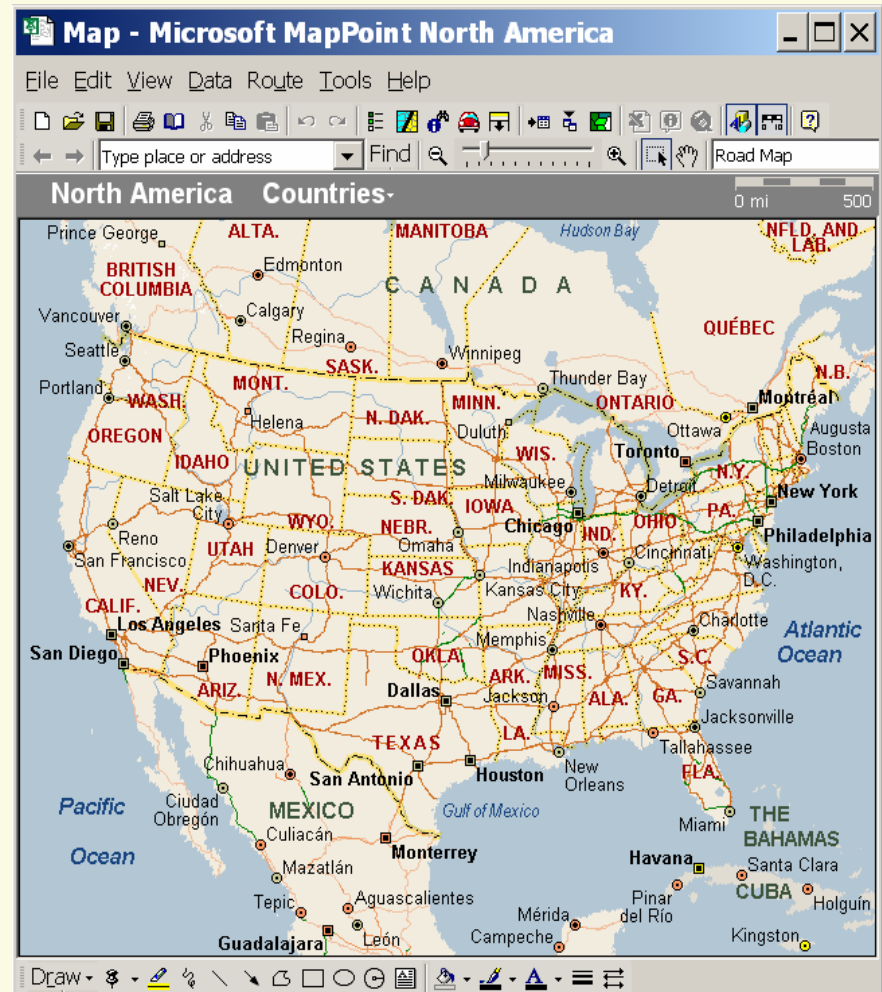
Procedure Editor - Run

Name	City	State

Address:

Add Update Distance Route Quit

Enter data or press ESC to end.




Return to sender

- ❏ The tloc temp-table is used to store each address entered and to store each location and pushpin com-handle associated with the address. The temp-table information is displayed in the browse and the frame.
- ❏ The findloc internal procedure is used to validate the address using the resultsquality property.
- ❏ The resstring variable contains the messages displayed to the user for each type of result.

Return to sender

- ❏ When a user presses enter on a particular row in the browse, mappoint zooms to that address using the goto method.
- ❏ The routing is calculated based upon the order that the rows were selected. The first row selected is the starting point and the second row is the ending point for the route.

Conclusion

-  There are several learning curves or knowledge bases needed in order to use OLE objects with a Progress application.
1. A thorough knowledge of Event Driven Programming in Progress
 2. A solid familiarity with the OLE Automation Server that you will be using, such as Word or Excel.

Conclusion

- ❏ There are several learning curves or knowledge bases needed in order to use OLE objects with a Progress application.
- 3. Make sure the VBA Help is loaded when installing MS Office and access it frequently. Use VBA's Object Browser for looking up constant values.

Conclusion

- ❏ There are several learning curves or knowledge bases needed in order to use OLE objects with a Progress application.
 - 4. Create macros to capture VBA commands that can be converted into Progress code.
Remember that the period (.) in VBA translates to a colon (:) in Progress.
 - 5. Use Progress' COM Object Viewer to see how the syntax is used in Progress.

Conclusion

- ❏ There are several learning curves or knowledge bases needed in order to use OLE objects with a Progress application.
- 6. Examine Progress' ActiveX examples stored in their src/samples/activex subdirectory.


Conclusion

❏ Like most things, with little documentation linking OLE Automation Servers to Progress clients, I found that trial-and-error was necessary to acquire knowledge.

Credits

- ❏ I would like to extend a special thanks to Don Sorcinelli at Progress Software Corporation who spent the time to show me where to look for things and elaborate on some of the thornier topics.
- ❏ His white paper “OLE Automation With MS Office97” I found to be very valuable. Please leave me your business card and I will be happy to e-mail the article to you.

Hey Excel, Analyze This!!!

 Now let's apply the same concepts to an Excel spreadsheet and chart.

Hey Excel, Analyze This!!!

 The procedure **ciexcel.p** looks similar to **cimerge.p**.

- The differences are that the user may select multiple items in the item browse for a customer to select on a graph.
- The other difference is a radio-set for chart type replaces the customer header information.

Hey Excel, Analyze This!!!


- ☐ Once the Excel button is selected, and provided that there is at least 1 customer selected and 1 item selected, the data is placed in a new workbook and worksheet and then charted.

Hey Excel, Analyze This!!!

`oWorkbook = oExcel.Workbooks.Add() .`

- ❏ A new workbook is created with the **Add** method for **Workbooks** collection of the Excel object.

Hey Excel, Analyze This!!!

 The first worksheet is accessed and named using the following:

```
oWSheet = oExcel:Sheets:Item(1).  
oWSheet.Name = "Customer Item  
Data".
```

Hey Excel, Analyze This!!!


```
if i = 1 then
```

```
assign
```

```
oWSheet:Columns("A"):ColumnWidth = 20
```

```
oWSheet:range("a1"):value = "Customer".
```

```
oWSheet:range("b1"):value = "Item".
```

 Next, the worksheet is formatted and populated with data by setting the **ColumnWidth** property for the columns and labels for the columns.

Hey Excel, Analyze This!!!

```
bcust:fetch-selected-row(i).
```

```
oWSheet:range("a" + string(i +  
2)):value = tcustomer.tname.
```

❏ The customer name is assigned to a given row.

Hey Excel, Analyze This!!!

assign


```
oWSheet:Columns(chr(97 +  
j)):ColumnWidth = 14
```

```
oWSheet:range(chr(97 + j) +  
"2"):value = titem.titem-name.
```

- ❏ Next the inner **DO** loop is used fill the rows with our customer sales data. Each column label is set to a specific item.

Hey Excel, Analyze This!!!

```
oWSheet:Range("B2:" + chr(97 +  
bitem:num-selected-rows) +  
string(bcust:num-selected-rows +  
2)):Select().
```

 **chr(97)** is the letter A. By adding the number of selected items in the browse to 97, we can determine the last lettered column we are using.

Hey Excel, Analyze This!!!


oExcel.Selection:

HorizontalAlignment = -4152.

 What does **-4152** mean?

- First, I created a macro to see how Visual Basic aligns items horizontally.
- Then, I had to go to the Object Browser to determine the value used of **XLRIGHT**, the constant for right horizontal alignment.

Hey Excel, Analyze This!!!

 The following is the Visual Basic code generated when you create the macro:

With Selection

.HorizontalAlignment = xlRight

.VerticalAlignment = xlBottom

.WrapText = False

.Orientation = 0

.ShrinkToFit = False

.MergeCells = False

End With

Hey Excel, Analyze This!!!

`oExcel.Selection.Style =
"Currency".`

- ❏ “Currency” is a format style for display numbers with dollar signs.
- ❏ Again, keep in mind that a range must be selected to set the **Style** property for the **Selection** object.


Hey Excel, Analyze This!!!

```
oChart:ChartWizard(oRange,  
    integer(charttype:screen-value),  
    1, 1, 1, 1, TRUE, "Customer Item  
Figures ", "Customer", "Value").
```

❏ Finally, **charttype**'s **SCREEN-VALUE** is used to set the **Gallery** parameter for the **ChartWizard** method.


- In addition, if the value of **charttype** is changed, it will instantaneously change the chart to the new type.

Hey Excel, Analyze This!!!

 Here are the parameters for the **ChartWizard** method.


1. Source - Handle to the range object being charted.
2. Gallery - Integer value specifying what kind of chart it will be.
3. Format - Specifies auto-formatting type to be applied.
4. Plot by - 1 is by rows, 2 is by columns.

Hey Excel, Analyze This!!!

 Here are the parameters for the **ChartWizard** method.

5. Category Labels - Specifies the number of rows and columns to contain category labels.
6. Series Labels - Specifies the number of rows or columns that contain series labels.
7. Has Legend - Logical specifying if there is a legend on the widget or not.
8. Title - Chart Title

Hey Excel, Analyze This!!!

 Here are the parameters for the **ChartWizard** method.

9. Category Title

10. Value and extra title.