Integrating a DataDirect Cloud Server into OpenEdge Applications

Peter Judge
Progress Software Corp.
August 2013

© Progress Software Corporation 2013
Document version 1.0

Contents

© Progress Software Corporation 2013

Introduction

OpenEdge applications are often deployed alongside other business systems that are responsible for key pieces of the overall business process. In these scenarios it is imperative that OE also connect (or integrate) with these other applications in order to execute the business process. DataDirect Cloud is a tool that facilitates this integration with SaaS applications like Salesforce, Microsoft Dynamics CRM, Oracle RightNow, and a number of other sources in the development pipeline. More information about DataDirect Cloud is available at http://www.datadirectcloud.com/why-datadirect-cloud.html .

The sample ABL shows how you can interact with the DataDirect Cloud (D2C) server, using a set of OOABL classes. These classes provide a simplified wrapper around a series of DLL calls to the D2C Driver via the ODBC Driver Manager. This simplification is primarily around removing the need to work directly with the ODBC API via memptrs and the like. The classes simplify the calls a consumer of the API must make in order to execute a SQL statement, gather corresponding metaschema and data results. By encapsulating the process, the ABL programmer needs only to formulate a SQL query and designate a target for its results while the classes handle all the handle allocations, state transitions and cleanup.

The API returns the data from the query executed (the result set) as either a JSON Object, or a dynamically-created temp-table. This temp-table is created from the column information of the query.

The sample code allows you to connect to any D2C data source, and to query it in any way you see fit.

Setup & environment

Prerequisites

- The API assumes that you are running a version of OE that has the same bitness as the underlying OS (ie 32-bit OE on 32-bit Windows). The default OpenEdge Windows install is a 32-bit install, although as of 11.3.0 there will be a 64-bit version. Trying to run 32-bit OpenEdge with the 64-bit drivers results in an error. There are some tweaks needed to install the 32-bit DLLs on a 64-bit OS, like Windows 7; these tweaks involve the registry settings and won't be described here.

- The fact that the ultimate datasources reside on the public internet somewhere means that you will need to be able to access such sites as DataDirectCloud.com and salesforce.com.

© Progress Software Corporation 2013

Setting up (cloud) data sources

There are a few setup steps required to be able to access cloud data sources.

Set up an account on salesforce.com or another supported cloud provider

This application is the ultimate data source - ie where the data actually lives. Some providers have time-limited accounts for trial purposes (eg Salesforce.com has a 30-day limited account).

⚠ Most of the examples in this document are based on a Salesforce.com datasource, since it was the first data source provided. The examples are also Windows-based (pertinent for setup more than runtime).

Create an account on the DataDirect Cloud site

Register for an account on http://www.DataDirectCloud.com.

Set up a cloud data source on DataDirect Cloud

In your DataDirectCloud account, create a new data source by selecting the Salesforce.com data source.

Setting	Value
Username	This will be your Salesforce.com login
Password	Your super-secret password.
Salesforce Login URL	`http://login.salesforce.com`
Security token	The Salesforce.com requires a token for remote access. This value **must** be specified in the Advanced tab.
Data Source Name	<any value>. This value will be used for the ODBC database name. For example, **`sfDSN`**

© Progress Software Corporation 2013

Download and install the D2C drivers

In DataDirectCloud, select the correct ODBC drivers for your system from the selection provided on the Downloads tab.

> The **ODBC Drivers Readme** talks about needing to install the Microsoft MDAC software. This is not necessary on Windows 7.

Set up a local data source

In the Windows **ODBC Administrator** tool, add a System DSN (or a User DSN, the setup is the same).

Setting	Notes
Data Source Name	<any value>. You can use something like **OED2C**
Database Name	The value specified above for the salesforce data source's name (see above). Eg, `sfDSN`
Authentication: User Name	Your user name (Progress ID) for DataDirectCloud
Authentication: Login Domain	<blank>
Data Source Authentication: Data Source User	<blank>
Data Source Authentication: Data Source Password	<blank>

Test the connection. You will be prompted for your password for your Progress ID, for the DataDirectCloud account.

Add test data

You will need to add some test data to the data source in order to be able to query it. Some data source may provide this test data for you.

© Progress Software Corporation 2013

Setting up the OpenEdge environment

Download the ABL API

> This sample was developed against OpenEdge release 11.3, and depends on a procedure library (PL) file shipped with that release.

The sample ABL API is available via the Samples page in Progress Developer Studio for OpenEdge (PDSOE). It can be accessed via the **Help > Samples** menu, and is also available via Progress Communities at http://www.psdn.com. This document also appears in the **doc** folder.

To manually create the project, use PDSOE's **File > Import** functionality and select the **abl_d2c.zip** archive. A project called `DataDirectCloud_ABL_API` will be created for you.

> Note that the project requires no local database, since all data will be retrieved from the Cloud data source

The project contains a Launch Configuration called `Run D2C Sample` which runs the sample procedure, `test/run_sample.p`. The code snippets in this document are taken from that procedure.

Dependencies

If you choose not to use PDSOE, you will need to manually change your propath in order to run the sample. On Windows, add `%DLC%/gui/rules/OpenEdge.BusinessRules.pl` to the propath. `%DLC%` represents the location of your OpenEdge 11.3 installation. If you would like to use this code in an AppServer environment, change the `'gui'` to `'tty'` in the propath entry.

Running the sample

The sample code provided connects to a D2C data source, and executes a couple of queries against it.

© Progress Software Corporation 2013

Connection

The first thing that the ABL needs to do is to connect to the D2C server. The connection is done using the `OpenEdge.Core.ServerConnection.IServerConnection` infrastructure created for connecting to a Business Rules server that was added to OE 11.3.

Configure the ABL with your username/etc

The connection information is contained in a file in JSON format, in the `cfg` folder in **`d2c.json`**. Replace the 'null' values below with the DataSourceName, user name and password you used to set up the System DSN above.

```
{ "DataSourceName"     : null,
  "UserName"           : null,
  "Password"           : null }
```

Connection code

The snippet below highlights the connection to the D2C server, via the `OpenEdge.Core.ServerConnection.ODBCConnection` object. This object is the interface through which we interact with the D2C server.

```
oConfig = cast(
    new ObjectModelParser():ParseFile('cfg/d2c.json'),
            JsonObject).
oD2CServer = new ODBCConnection(oConfig).
oD2CServer:Initialize().
```

Once we've connected to the server, and initialised it, we can query the data within.

Direct query execution

The D2C connection now allows us to perform SQL queries against the data source defined in D2C. The queries shown here are against Salesforce.com; obviously the tables and fields queried will be specific to each data source.

© Progress Software Corporation 2013

Retrieving schema information

There are 2 methods on the `OpenEdge.Core.ServerConnection.ODBCConnection` type for getting table and related schema information from the data source. These are **`GetTableSchema`** and **`GetTables`**. The former is for a single table, the latter for all tables. More detail appears later in this document.

It is not necessary to retrieve the schema in order to execute a query (ie if you already know the table/field names).

Returning a temp-table

The API will execute the passed-in SQL statement, using the query's result set to create and populate an ABL temp-table named `resultset`. The temp-table will contain a field for each column in the query, and a record for each row in the SQL result set that is returned.

Currently the API is limited to returning a dynamically-created temp-table (as opposed to populating a pre-existing temp-table). This means you need to use dynamic queries etc to work with the data.

```
/* Execute a SQL SELECT statement and get the result set as an ABL
temp-table */
cStmt = " select USERNAME, LASTNAME, EMAIL from USER ".

oD2CServer:ExecuteStatement(
             input cStmt,
             output table-handle hResultSet).

/* just dump the output to disk. Obviously you would do more with
this data in the real world */
hResultSet:write-json(
           'file', 'temp/resultset-table.json', true).
```


© Progress Software Corporation 2013

Returning JSON-formatted data

The API will execute the passed-in SQL statement, and return an ABL **`JsonObject`** based on the results of that query. This JSON data will contain both the schema information about the fields in the query, and also the data returned in the SQL result set.

```
/* Execute a SQL SELECT statement and get the result set in JSON
form */
cStmt = "select ACCOUNTNUMBER, SYS_NAME, ANNUALREVENUE,
NUMBEROFEMPLOYEES, DESCRIPTION, SLAEXPIRATIONDATE from ACCOUNT ".

oD2CServer:ExecuteStatement(input cStmt, output oResultSet).

/* just dump the output to disk. Obviously you would do more with
this data in the real world */
if valid-object(oResultSet) then
    oResultSet:WriteFile('temp/resultset-json.json', true).
```

The JSON object returned has 3 properties: `query` (the query that was executed), `columns` (schema information about the columns used by the query) and `resultset` (the data returned). The (truncated) output from the query about is shown below

© Progress Software Corporation 2013

JSON result set


```
{"query"         : "select ACCOUNTNUMBER, SYS_NAME, ANNUALREVENUE,
           NUMBEROFEMPLOYEES, DESCRIPTION, SLAEXPIRATIONDATE from
           ACCOUNT ",
  "columns"      : [
     {"ColumnNum"      : 1,
      "ColumnName"     : "ACCOUNTNUMBER",
      "CType"          : 1,
      "CTypeSize"      : 1025,
      "AblType"        : "CHARACTER",
      "NumDecimals"    : 0,
      "IsNullable"     : true,
      "ColMaxWidth"    : 40
     },
     {"ColumnNum"      : 2,
      "ColumnName"     : "SYS_NAME",
      "CType"          : 1,
      "CTypeSize"      : 1025,
      "AblType"        : "CHARACTER",
      "NumDecimals"    : 0,
      "IsNullable"     : false,
      "ColMaxWidth"    : 255
     },
     {"ColumnNum"      : 3,
      "ColumnName"     : "ANNUALREVENUE",
      "CType"          : 8,
      "CTypeSize"      : 8,
      "AblType"        : "DECIMAL",
      "NumDecimals"    : 0,
      "IsNullable"     : true,
      "ColMaxWidth"    : 53}
   /* etc */
   ],
  "resultset"    : [
     {"ACCOUNTNUMBER"       : "CC978213",
      "SYS_NAME"            : "GenePoint",
      "ANNUALREVENUE"       : 30000000.0,
      "NUMBEROFEMPLOYEES"   : 265,
      "DESCRIPTION"         : "Genomics company engaged in mapping
                               and sequencing of the human genome
                               and developing gene-based drugs",
      "SLAEXPIRATIONDATE"   : "2013-01-14"}
    /* etc */
    ]
}
```


© Progress Software Corporation 2013

Other operations

Getting datasource-supported data types

There are multiple SQL data types that are supported by D2C. This sample has only been tested against those used by Salesforce.com (although since we use SQL Concise types, many other data sources should be supported too).

To see what types are supported by your datasource, call the **`GetTypeInfo()`** method on the **`OpenEdge.Data.ODBC.SqlGetTypeInfo`** class. See example below

```
define variable oResultArray as JsonArray no-undo.

oResultArray = oD2CServer:GetTypeInfo().
if valid-object(oResultArray) then
   oResultArray:WriteFile('temp/type-info-all.json', true).
```

Method Name	Return Type	Access	Parameters	Comments
GetTypeInfo	`Progress.Json.ObjectModel.JsonArray`	public		Returns an array of JSON Objects describing the data types supported by the data source.
GetTypeInfo	`Progress.Json.ObjectModel.JsonArray`	public	input `OpenEdge.Data.ODBC.SqlTypeEnum`	Returns a filtered array of JSON Objects describing the data types supported by the data source.

© Progress Software Corporation 2013

Getting schema information

The `ODBCConnection` object also provides an API for querying the data source about the tables and fields (columns) it contains.

Method Name	Return Type	Access	Parameters	Comments
GetTables	`Progress.Json.ObjectModel.JsonObject`	public	input plChildSchema as logical	Returns schema information for all tables in the data source. If the parameter is true, all child schema elements (initially only columns) are also returned
GetColumns	`Progress.Json.ObjectModel.JsonObject`	public	input pcTableName as character	Returns schema information about all columns for the specified table.
GetTableSchema	`Progress.Json.ObjectModel.JsonObject`	public	input pcTableName as character	Returns table and column schema information for the specified table. Differs from `GetColumns` in that table schema is also included

A truncated example of the output produced by `GetTableSchema` for the Salesforce USER table is below.

© Progress Software Corporation 2013

Table and column schema

An example of table and column schema is shown below.

```
{"query"    : "SQLTables",
 "tables"   : [
  {"TABLE_CAT"   : null,
   "TABLE_SCHEM" : "SFORCE",
   "TABLE_NAME"  : "USER",
   "TABLE_TYPE"  : "TABLE",
   "REMARKS"     : null,
   "columns"     : [
    { "TABLE_CAT"           : null,
      "TABLE_SCHEM"         : "SFORCE",
      "TABLE_NAME"          : "USER",
      "COLUMN_NAME"         : "ROWID",
      "DATA_TYPE"           : -9,
      "TYPE_NAME"           : "ID",
      "COLUMN_SIZE"         : 18,
      "BUFFER_LENGTH"       : 54,
      "DECIMAL_DIGITS"      : null,
      "NUM_PREC_RADIX"      : null,
      "NULLABLE"            : 0,
      "REMARKS"             : null,
      "COLUMN_DEF"          : null,
      "SQL_DATA_TYPE"       : -9,
      "SQL_DATETIME_SUB"    : null,
      "CHAR_OCTET_LENGTH"   : 54,
      "ORDINAL_POSITION"    : 1,
      "IS_NULLABLE"         : "N"   },
    /* etc */
   ]
 }]
}
```

- The `OpenEdge.Data.ODBC.SqlGetTables` class contains a detailed description of the schema for columns in its static constructor.
- The `OpenEdge.Data.ODBC.SqlGetColumns` class contains a detailed description of the schema for tables in its static constructor.

Conclusion

This document describes how to connect your OpenEdge application to the DataDirect Cloud service - and thus gain access to a variety of Cloud-based data

© Progress Software Corporation 2013

sources - using a simple ABL API. This functionality allows you to use the data from those Cloud-based datasources to enrich your OpenEdge application.

The associated project code is also available on Progress Communities, on the Integrating a DataDirect Cloud Server into OpenEdge Applications page. Please feel free to provide comments and feedback, and questions you might have to that page and its parent community.

© Progress Software Corporation 2013