Overview

The OLE Automation class enables you to invoke and communicate with an external application that supports OLE automation as a server. In this communication, SAS acts as a client while the automation application acts as a server. An OLE automation object can send methods to the automation server for execution. The OLE automation object can also get and set property values through the automation server. This class is a nonvisual class although the automation application may provide visuals outside of the context of SAS/AF software.

Note: The OLE Automation class is available only on operating systems that support OLE, such as Microsoft Windows. For complete information about using OLE automation in your SAS/AF application, see your host companion.

Parent:

sashelp.fsp.object.class

Class:

sashelp.fsp.HAuto.class

Using the OLE Automation Class

Populating a Microsoft Excel Spreadsheet with SAS Data

The following example uses SCL code to populate a Microsoft Excel spreadsheet with data from a SAS data set:

Load an instance of the OLE Automation class and invoke Microsoft Excel as the new object. Set the object's Visible attribute to true so you can see the automation.
LAUNCHXL:
   hostcl = loadclass(‘sashelp.fsp.hauto’);
   call send (hostcl, ‘_new’, excelobj, 0,
      ‘Excel.Application.5’);
   call send (excelobj,’_setProperty’,
      ‘Visible’,’True’);
   return;

Get the identifier for the current Workbooks property and add a worksheet to it. Then get the
identifier for the new worksheet.

CREATEWS:
   call send (excelobj,’_getProperty’,
      ‘Workbooks’,
      wbsobj);
   call send (wbsobj, ‘_do’, ‘Add’ );
   call send (excelobj,’_getProperty’,
      ‘ActiveSheet’,
      wsobj );

Open a SAS data set.

   dsid=open(‘sasuser.class’,’i’);
   call set(dsid);
   fetch(dsid);
   nvar=attrn(dsid, ‘NVARS’);
   nobs=attrn(dsid, ‘NOBS’);

Traverse the data set and populate the cells of the Excel worksheet with its data, row by row.

   do col=1 to nvar;
      call send (wsobj, ‘_compute’, ‘Cells’,1,
         col,retcell);
      var=varname(dsid,col);
      call send (retcell, ‘_setProperty’,
         ‘Value’,var);
   end;
   do while (rc ne -1);
      do row = 1 to nobs;
         do col = 1 to nvar;
            r=row+1;
            call send (wsobj, ‘_compute’,
               ‘Cells’, r ,col,
               retcell);
            if vartype(dsid,col) eq ’N’ then
               var=getvarn(dsid,col);
            else var=getvarc(dsid,col);
         end;
      end;
   end;
call send (retcell, '_setProperty', 'Value', var);
end;
fetch(dsid);
end;
end;
return;

Close the worksheet and end the Excel session.

QUITXL:
call send (excelobj, '_getProperty', 'ActiveWorkbook', awbobj);
call send (awbobj, '_do', 'Close', 'False');
call send (excelobj, '_do', 'Quit');
call send (excelobj, '_term');
return;

As you can see from this example, automating an application object requires some knowledge of the object's properties and methods. To help you decide which SCL commands to use for an Excel automation object, you can use the Macro Recorder in Excel to perform the task you want to automate and then look at the Visual Basic code that is generated. It is then relatively simple to map the Visual Basic code to comparable SCL statements and functions.

Here are some excerpts of Visual Basic code with their SCL equivalents:

<table>
<thead>
<tr>
<th>Visual Basic code</th>
<th>OLE Automation in SCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Launch Excel and make it visible</td>
<td>hostcl = loadclass('sashelp.fsp.hauto');</td>
</tr>
<tr>
<td>Set excelobj = CreateObject(&quot;Excel.Application.5&quot;)</td>
<td>call send (hostcl, '_new', excelobj, 0, 'Excel.Application.5')</td>
</tr>
<tr>
<td>excelobj.Visible = True</td>
<td>call send (excelobj, '_setProperty', 'Visible', 'True');</td>
</tr>
<tr>
<td>Create a new worksheet</td>
<td>call send(excelobj, '_getProperty', 'Workbooks', wbsobj);</td>
</tr>
<tr>
<td>Dim wbsobj, wsobj As Object</td>
<td>call send(wbsobj, '_do', 'Add');</td>
</tr>
<tr>
<td>Set wbsobj = excelobj.Workbooks</td>
<td>call send(excelobj, '_getProperty', 'ActiveSheet', wsobj);</td>
</tr>
<tr>
<td>wbsobj.Add</td>
<td>call send(excelobj, '_getProperty', 'ActiveSheet', wsobj);</td>
</tr>
<tr>
<td>Set wsobj = excelobj.ActiveSheet</td>
<td>call send(excelobj, '_getProperty', 'ActiveSheet', wsobj);</td>
</tr>
</tbody>
</table>
Visual Basic code

Set the value of a cell

wsobj.Cells(row + 1, col).Value = var

\[ r = \text{row} + 1; \]
\[ \text{call send(wsobj,'_compute','Cells', r, col, retcell);} \]
\[ \text{call send(retcell,'_setProperty','Value',var);} \]

Close the Excel application object

excobj.ActiveWorkbook.Close ("False")
excobj.Quit

\[ \text{call send(excobj,'_getProperty','ActiveWorkbook', awbobj);} \]
\[ \text{call send(awbobj,'_do','Close', 'False');} \]
\[ \text{call send(excobj,'_do','Quit');} \]
\[ \text{call send(excobj,'_term');} \]

Methods

Methods specific to the OLE Automation class are described here. Inherited methods are described in the Object class.

Dictionary

__compute

Invokes an OLE-supplied method that returns a value

Syntax

CALL SEND (OLE-object, '_compute', in-OLE-method<, in-parm,...in-parm, out-value>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-OLE-method</td>
<td>C</td>
<td>specifies the OLE-supplied method name</td>
</tr>
<tr>
<td>in-parm</td>
<td>C</td>
<td>N</td>
</tr>
<tr>
<td>out-value</td>
<td>C</td>
<td>N</td>
</tr>
</tbody>
</table>

Details

_computes sends an OLE-supplied method and its parameters to the automation server for execution by the server application. The number of parameters (in-parm arguments) needed varies among different OLE objects and methods. Only OLE-supplied methods that have a return value should be used with the _compute method. For OLE-supplied methods with no return values, use the _do method.
**Example**

The following example sets the cell value for row 2, column 5 of a spreadsheet OLE automation object to 100:

```plaintext
call send(oleobj, '_compute', 'Cells', 2, 5, cellobj);
call send(cellobj, '_setProperty', 'Value', 100);
```

---

**_do**

Invokes an OLE-supplied method that has no return value

---

**Syntax**

CALL SEND (OLE-object, '_do', in-OLE-method<, in-parm,...in-parm>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-OLE-method</td>
<td>C</td>
<td>specifies the OLE-supplied method name</td>
</tr>
<tr>
<td>in-parm</td>
<td>C</td>
<td>N provides a parameter to the OLE-supplied method</td>
</tr>
</tbody>
</table>

**Details**

_do sends an OLE-supplied method and its parameters to the automation server for execution by the server application. The number of parameters (in-parm arguments) needed varies among different OLE objects and methods. Only OLE-supplied methods that have no return value should be used with the _do method. For OLE-supplied methods with return values, use the _compute method.

**Example**

The following statement invokes the 'Quit' method of the automation application:

```plaintext
call send(oleobj, '_do', 'Quit');
```

---

**_getProperty**

Returns the value of a property from an OLE automation server

---

**Syntax**

CALL SEND (OLE-object, '_getProperty', in-OLE-property, out-value);
### _getReferenceId

Returns a reference identifier for use with any OLE-supplied method that requires an OLE object as one of its parameters.

**Syntax**

CALL SEND (OLE-object, '_getReferenceId', out-refid);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>out-refid</td>
<td>C</td>
<td>contains the returned reference identifier</td>
</tr>
</tbody>
</table>

**Details**

_getReferenceId gets the OLE object identifier. The value returned is used in subsequent method calls where the OLE-supplied method requires an OLE object as one of its parameters. This value should be used for the object parameter.

**Example**

The following example returns the reference identifier for the OLE automation object. This identifier is then sent as a parameter value to an OLE-supplied method requiring an object identifier.

```plaintext
  call send(oleobj1, '_getReferenceId', refid);
  call send(oleobj2, '_do', 'NewChart', refid, p1, p2);
```

### _inError

Returns an object's ERROR status.
**Syntax**
CALL SEND (OLE-instance, '_inError', error-status<, error-msg>);

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error-status</td>
<td>N</td>
<td>returns a value indicating whether an automation error has been encountered for the object</td>
</tr>
<tr>
<td>error-msg</td>
<td>C</td>
<td>returns the automation error message</td>
</tr>
</tbody>
</table>

**Details**
_inError can detect errors encountered from automation calls. _inError returns the status of the last automation call and should be called prior to any subsequent automation calls.

**Example**

The following example detects that an error was encountered during the previous _getProperty call:

```plaintext
call send(objid,'_getProperty', 'ActiveObject', actobj);
call send(objid,'_inError',inerror, errmsg);
if inerror then
  link handle_err;
```

**_new**

Creates a new instance of an OLE automation object

**Syntax**
CALL SEND (OLE-instance, '_new', new-ole-id, init-arg, ole-auto-app);
### Example

This example creates a new OLE automation object and assigns the SCL identifier `exclauto` to the new object:

```plaintext
hostcl=loadclass('sashelp.fsp.hauto');
call send (hostcl, '_new', exclauto, 0, 'Excel.Application.5');
```

### _setProperty

Assigns a value to a property of an OLE automation server

#### Syntax

`CALL SEND (OLE-object, '_setProperty', in-OLE-property, in-value);`

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-OLE-property</td>
<td>C</td>
<td>specifies the OLE property name</td>
</tr>
<tr>
<td>in-value</td>
<td>C</td>
<td>N</td>
</tr>
</tbody>
</table>

#### Details

 SetProperty assigns a value to a property of an OLE automation server.

#### Example

The following example sets the value of the Visible property for an OLE automation object:

```plaintext
call send(oleobj, '_setProperty', 'Visible', 'True');
```