OLE and the SAS® System for Windows Release 6.11

Jennifer Clegg and Carol Rigsbee, SAS Institute Inc., Cary, NC

ABSTRACT

This paper describes the support for OLE (object linking and embedding) provided in the SAS System for Windows Release 6.11. Features include support for embedded and linked objects, drag and drop, visual editing, OLE automation, and OLE controls. OLE support within the SAS System is available in SAS/AF® FRAME entries and SAS/EIS® applications.

INTRODUCTION

This paper provides an overview of the OLE support in the SAS System. For more details on OLE in general, see the documentation for the Microsoft Windows operating environment. For further details on OLE support within the SAS System, refer to the *SAS Companion for the Microsoft Windows Environment, Second Edition.*

OLE is an application integration technology available on Microsoft Windows platforms. It provides a means of integrating multiple heterogeneous sources of information, or objects, into one homogenous document. These objects can include text, graphics, sound, video clips, and more. The applications that provide these objects are OLE servers. The applications that contain these objects are OLE containers.

Since Release 6.08, the SAS System has provided support for OLE 1.0 as a container. OLE 1.0 provided the ability to create embedded and linked objects. OLE 2.0 extends this feature set to include drag and drop, visual editing, OLE automation, and OLE controls (OCXs). Since OLE 2.0 completely encompasses OLE 1.0 functionality, the SAS System continues to interoperate with OLE 1.0 servers.

OLE functionality can be categorized in three main areas: basic container support, OLE automation support, and OLE controls support. The SAS System supports all three areas. Each section of this paper provides an explanation of the particular OLE terminology followed by specific information about the use of OLE within the SAS System.

BASIC CONTAINER SUPPORT

Overview of OLE

The SAS System's basic container supports the creation of embedded and linked objects, drag and drop, and visual editing. Drag and drop support and visual editing are both new with OLE 2.0.

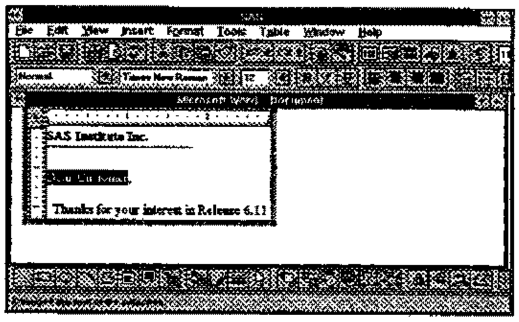
Embedded objects physically reside within the container document. The document provides the disk-based storage for the object. To distribute a document containing an Embedded object, you simply copy the document.

For Linked objects the source of the data physically resides where it was initially created. The container's storage for the object contains a pointer to the physical data on disk, such as a file. To distribute a document containing a Linked object, you must copy the document as well as the file that is the source of the link.

For both Embedded and Linked objects, the object's storage within the container includes a static representation, typically a Metafile, of the object. This static representation allows you to view an object if the server or link source is unavailable. The server provides object creation and editing for Embedded and Linked objects.

One of the new features provided with OLE 2.0 is the ability to drag and drop objects onto containers. Drag and drop supports keyboard modifiers that you can use to change the behavior of the drop. By default, dragging an object moves the object from the server application into the container application. If you want to copy an object, hold down the Ctrl key when you drop the object in the target application. The cursor will change to an arrow with a box and a plus (+) sign. To create a link to the object data, hold down the Ctrl and Shift keys when you perform the drop. The cursor will become an arrow with a box and an equal (=) sign. If the target area is not a valid drop site or the operation is not supported, the cursor will change to the not (∅) sign.

Another new feature of OLE 2.0 is visual editing. It allows you to edit an object within the context of its container. The container takes on the user interface of the server. The menus, toolbars, and status line switch to the ones normally provided by the server. Below is an example of a Microsoft Word object being visually edited in the SAS container.



SAS Institute Inc.

Thanks for your interest in Release 6.11

With OLE 1.0 open editing was the only supported mechanism for editing. In open editing the server is launched as a separate application and all editing is performed in a separate window.

By design, Linked objects only support open editing. Embedded objects may support visual editing or open editing depending on the server. Additionally, a server that supports visual editing will most likely support open editing as well.

Overview of OLE and the SAS System
There are multiple ways to create Embedded and Linked objects with the SAS System. First, you need to bring up the SAS BUILD:DISPLAY window so you can create a FRAME entry that contains an OLE object. To do this submit the following statement:

```
proc build c=sasuser.ole.demo.frame; run;
```

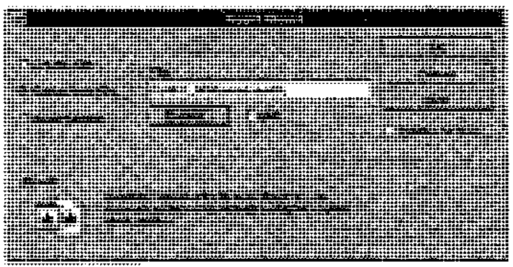
To create OLE objects you can use the MAKE or FILL menu selection. To access these menu selections, click the right mouse button while your mouse is positioned in the BUILD window. You could alternatively drag out a region, click the right mouse button within this region to access the menu, and select FILL. If you use MAKE, the OLE object size will be defined based on server defaults. With FILL, the object created will be the size of the region you defined. For bitmaps and other objects that do not scale well, MAKE is preferable because the object retains its appearance.

For Embedded and Linked objects in the SAS System, HSERVICE entries are created in SAS Catalogs to store the necessary information about objects. These entries are only portable to other Windows platforms.

Creating OLE Objects in FRAME
One of the ways you can create OLE objects in FRAME is to use the Insert Object dialog. To access this dialog click on the MAKE menu item from the popup menu. Scroll through the list of objects and select `OLE - Insert Object`. The dialog is shown below.

Notice that there are three radio buttons in this dialog. The first one is `Create New`. When this radio button is selected, you will see a list of available object types. This list will vary based on the OLE-capable applications that exist on your machine. If you select one of these objects, an object of that type will be created in the frame. Objects created in this way are always Embedded. Since this is a new instance of an object, you will automatically be placed into edit mode, either visual editing or open editing, depending on what the server supports. Click outside the object, elsewhere on the frame, to end visual editing. Exit the server to end open editing.

If you select the second radio button, `Create from File`, you will be prompted to enter a filename. The Insert Object dialog with this option selected is shown below.



If you do not know the name of the file, you can browse for the file that you want to use. Using this method will create an Embedded object based on the contents of the file. To create a Linked object, click the `Link` check box to select it. Editing will not be automatically initiated in this instance since the object you are creating already exists and has data.

Discussion of the third radio button, `Insert Control`, is described later in the OLE Controls section.

Another way to create an Embedded or Linked object is to use the Paste Special dialog. This dialog is also available from the MAKE menu as `OLE - Paste Special`.

The Paste Special dialog allows you to paste an object based on data from the clipboard. You may create objects of the following types: Embedded, Linked, Metafile, Device Independent Bitmap, or Bitmap. The choices available will be determined by the formats placed on the clipboard by the server. Select the desired choice and click the `Paste` button to create any type of object except Linked. You must select the `Paste Link` button to create a Linked object. Metafile, Device Independent Bitmap, and Bitmap are all static data representations of an object. They can only be viewed in a container, not edited, and are a good way to create fancy pictures in your application.

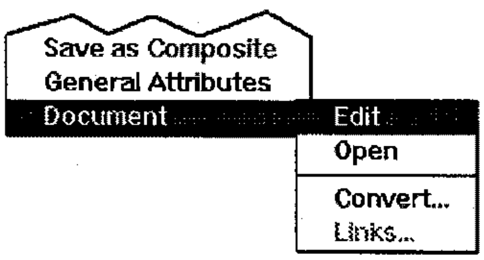
Another way to create an object is to read an existing object from a SAS catalog. To read an object from a catalog select `OLE - Read Object` from the `MAKE` menu. When you choose this option, you will be prompted for the HSERVICE entry name of the object you want to use. Reading an object from a catalog is equivalent to copying an object. However, this method allows you to access objects in other frames. You must change the name of the HSERVICE entry in the Object Attributes dialog after you create the object if you want this object to be saved as a separate catalog entry.

You can also use drag and drop to create a new object based on an existing object or data in another application. Through drag and drop you create a new object by dragging an object from an OLE-capable application and dropping it on the frame.

OLE Verbs

One important feature of OLE objects is that they expose verbs. Verbs are actions that can be performed on an object. Each OLE object has a default verb associated with it. Most objects expose "edit" as the default verb. Some objects, such as media clip, expose "play" as the default verb. The default verb is important because it is the action that is performed when you double-click an object in the frame. Most objects have more than one verb.

To access all the verbs for an object in BUILD, click on the object with the right mouse button to access its menu. The name of the OLE object is the last item on the menu. This menu contains a list of selections available with this object. Below is a fragment of the MAKE menu showing the object's menu.



The first set of selections (before the separator) is the list of verbs. The first selection is the default verb. You can select any of these verbs and the action will be performed. Additional selections available from this menu include the Links and Convert dialogs.

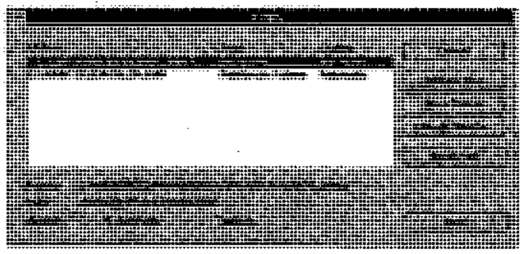
You can also view the list of verbs for an object by invoking the Object Attributes dialog and selecting the Associated Verbs item. Using SCL, you can programmatically execute verbs using the

EXECUTE method. An example is shown below:

```
call notify('oleobj','_EXECUTE_','Edit');
```

The Links and Convert Dialogs

The Links dialog provides important functionality for the maintenance of OLE Linked objects. One particular function is the ability to repair broken links. Broken links may occur when you move a file that is the link source for a container object. With the Links dialog you can change the location of the physical file that the link refers to, thus repairing the link. In the Links dialog you can also force an update of a link, edit the source of a link, or break a link. Breaking the link will change the object to a static representation. The Links dialog displays all Linked objects in the current frame. You can access the Links dialog by typing DLGLINKS on a command line or by using SCL to pass DLGLINKS as a verb to the object using the _EXECUTE_ method. In BUILD you can also access the Links dialog from the object's menu.



The Convert dialog allows you to specify an alternate editor for an object when the original server is unavailable. If you convert an object, it permanently becomes an object of the new type and its storage is rewritten in the new object format. You can also use this dialog to activate an object as an object of another type. The storage for the object is unaltered and all objects of this type on your system will now use the new application for editing. Conversion/activation support depends on the servers installed on your system. You can access the Convert dialog by typing DLGCONVERT on a command line. When accessing the Convert dialog in this way, an OLE object must be selected in the frame. You also can use SCL to pass DLGCONVERT as a verb to the object. Additionally, in BUILD, you can access the Convert dialog from the object's menu.

OLE AUTOMATION

Automation is a way to manipulate a application from outside that application. It effectively allows you to programmatically control, or script, an application. An OLE Automation Server is an application that exposes objects for scripting. These objects are OLE Automation Objects. Applications that access these objects are OLE Automation Controllers. Controllers provide a mechanism such as a script language to access automation objects.

You can control OLE Automation Objects with methods and properties. Methods are functions that perform actions on objects. Properties are function pairs that set or get information about the state of the object. The server's documentation describes the objects and their properties and methods.

The SAS System functions both as an automation server and an automation controller. The server exposes an automation object to allow access to the command line functionality of the SAS System. Any automation controller, such as Visual Basic, can access this object.

The automation object has the following methods and properties:

```
PushCommand (method)
        pushes a command to the command
        line of the active window of the
        SAS System session
Quit (method)
        ends the current SAS System
        session
Visible (property)
        shows/hides the SAS System
        session
Title (property)
        changes the title of the SAS
        System session
```

The SAS System installation process updates the registration database with the necessary information for the automation server. To start the server from an automation controller, you need its ProgID (programmatic identifier). The identifier for the server is "SAS.Application.61102". The automation server support is available with the BASE SAS software. For complete and final details on the automation server, please see the online help or refer to the paper, "The SAS System OLE Automation Server".

As an OLE Automation Controller, the SAS System must provide a way to access OLE Automation Objects. This access is provided through the following SCL methods:

```
_NEW_
        creates an OLE Automation object
_SET_PROPERTY_
        sets the value of a property
_GET_PROPERTY_
        gets the value of a property
_DO_
        invokes method with no return
        value
_COMPUTE_
        invokes method with return value
_GET_REFERENCE_ID_
        returns SCL reference identifier
        for the object
```

An SCL example which uses these methods follows. This example invokes Microsoft Excel, opens a worksheet, selects a range of cells, and charts them.

```
length rangeid $80;

/* create excel automation object */
hostcl = loadclass('sashelp.fsp.hauto');
call send (hostcl, '_NEW_', excelobj,
  0, 'Excel.Application.5');
call send (excelobj, '_SET_PROPERTY_',
  'Visible','True');

/* open worksheet */
call send(excelobj, '_GET_PROPERTY_',
  'Workbooks', wbsobj);
call send(wbsobj, '_DO_', 'Open',
  'sales.xls');
call send(excelobj, '_GET_PROPERTY_',
  'ActiveSheet', wsobj);

/* create a chart object */
call send(wsobj, '_COMPUTE_'
  'ChartObjects' chobjs );
call send ( chobjs, '_DO_', 'Add', 144,
  13.5, 287.25, 240.75 );
call send ( chobjs, '_DO_', 'Select' );
call send(excelobj, '_GET_PROPERTY_',
  'ActiveChart', chartobj );

/* get the range of cells to chart */
call send(wsobj, '_COMPUTE_', 'Range',
  'C5', 'D8', rangeobj );
call send(rangeobj, '_GET_REFERENCE_ID_',
  rangeid);

/* chart the cells */
call send (chartobj, '_DO_',
  'ChartWizard', rangeid, -4098, 6,
  1, 0, 0, 1, 'Automation Demo!',
  'Column', 'Value', 'Row' );
```

You can also automate Embedded or Linked objects that exist in your FRAME by using these same SCL methods.

Many applications support Visual Basic to automate OLE servers. For example, you can use the macro recorder in Excel to capture the Visual Basic commands necessary to perform some functions. This code can be quickly translated into SCL. See the Appendix for a comparison of SCL and Visual Basic commands.

OLE CONTROLS

OLE controls (OCXs) evolved based on technology first provided in Visual Basic with VBXs. Third party vendors provide

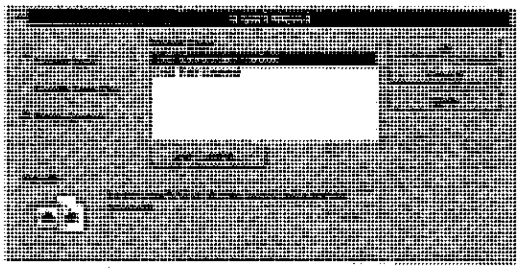
VBXs to extend the control set of Visual Basic. The OLE developers thought VBXs were a good idea but felt their functionality could be enhanced. This extended design resulted in OCXs, named for the OLE controls' three letter DOS extension. Current releases of Visual Basic provide support for OCXs and vendors are currently converting their VBXs into OCXs.

OCXs are like embedded OLE objects with added functionality. The main additions are support for ambient properties and events. Ambient properties allow communication between a container and an OCX for notification of property changes such as fonts and colors. For example, if the font in the container changes, the container will notify the OCX so it can change its font if applicable. Events are actions that are generated by the OCX to the container. For example, a push button control might generate a click event.

You can use any available 32-bit OCX in FRAME. Since the SAS System provides a 32-bit container, all OCXs used in the SAS System must also be 32-bit. Additionally, we are providing two OCXs: a text entry and a combo box. These are two of the most often requested controls.

Like Embedded and Linked objects, the object information for an OCX is stored in an HSERVICE entry within a SAS catalog. When distributing a catalog that contains an OCX, you must copy the catalog as well as the OCX.

To use an OCX in the frame, you use the Insert Object dialog that was discussed earlier in the basic container section. From within this dialog select the `Insert Control` radio button. The dialog with this option selected is shown below.



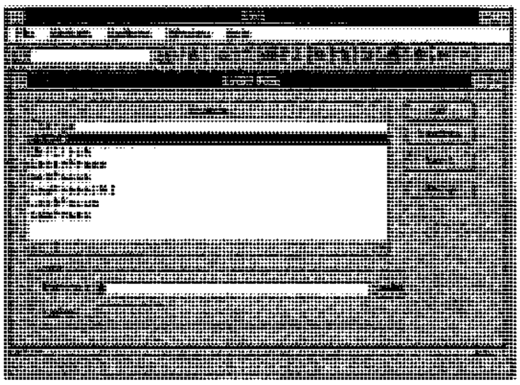
The list of controls displayed will vary based on the controls installed on your machine. If the control you wish to use is not shown, it is not currently registered on your system. To register it, select the `Add Control` button to browse and find the OCX you want to use. After clicking OK, the control will be added to the OCX list. Select the control that you wish to use and click `OK` to create an instance of it in the frame. Alternatively, you can drag and drop a control in the frame from another application or paste one in from the clipboard using `OLE - Paste Special`.

OCXs provide a property sheet for access to some of the control's properties. To access the property sheet in BUILD, select the Properties verb from the object's menu. To access the property sheet using SCL, you send the Properties verb to the object using the _EXECUTE_ method. Properties is usually the default verb for a control allowing you to double-click the mouse within the object's border to display the property sheet.

To access all the properties and methods of a control, you use a subset of the SCL methods provided for automation. The SCL methods _GET_PROPERTY_ and

_SET_PROPERTY_ access the properties of a control. The _DO_ and _COMPUTE_ SCL methods invoke methods of a control. You must know the names of the properties and methods for the object as well as any arguments. This information should be provided with the documentation for the OCX.

OCXs generate events that you can respond to in your SCL code. In the SAS System you can view and map these events using the Event Map dialog. This dialog is show below.

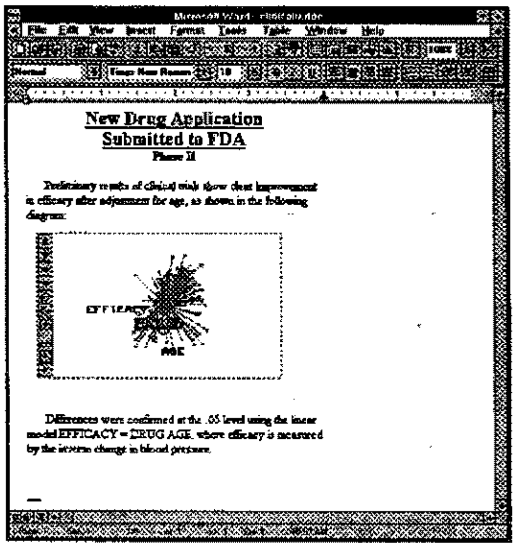


To access this dialog invoke the Object Attributes dialog and select Event Map. You may now select an event and specify the name of the SCL, FRAME, or PROGRAM source entry and (if applicable) the SCL label where the event-handling code resides. You can edit the SCL for the event from this screen as well. When you map an event, you must compile the SCL outside of the frame in which the control exists.

You may have experienced problems accessing the SAS menus in BUILD when clicking on an OCX. For this reason we have added a $OLEUIDEAD option. This option disables the control's user interface in BUILD. When using this option, clicks on the OCX go to the container for access to menus. If you use this option, double-clicking an object in BUILD will not execute the default verb. You must execute any verbs from the object's menu. Most verbs you execute on the object will activate the object, enabling its user interface. This allows you to interact with the object in BUILD if desired. Clicking outside the object will once again disable its user interface.

FUTURE DIRECTIONS

As technology continues to advance, we are constantly examining and researching future directions. For example, we are currently researching an OLE server implementation by prototyping SAS/Insight® as an OLE object. The following screen dump shows an Insight rotating plot embedded within a Microsoft Word document. Double clicking on the graph activates the object in place, allowing you to modify the graph. This is a sample of possible functionality provided by an OLE Insight object.



New Drug Application
Submitted to FDA
Phase II

Preliminary results of clinical trials show clear improvement in efficacy after adjustment for age, as shown in the following diagram:

Differences were confirmed at the .05 level using the linear model EFFICACY = DRUG AGE, where efficacy is measured by the [illegible] change in blood pressure.

This is just one of many ongoing research efforts to help advance the future versions of the SAS System.

CONCLUSIONS

This paper summarizes the support for OLE 2.0 within the SAS System for Windows Release 6.11. New features include support for drag and drop, visual editing, OLE automation, and OLE controls. OLE support greatly extends the capabilities of SAS/AF software by allowing third party developed-objects to be integrated with your SAS/AF application.

REFERENCES

Corcoran, Brian (1996), "The SAS System OLE Automation Server", Cary, North Carolina: SAS Institute, Inc.

Jennifer Clegg
SAS Institute, Inc.
100 SAS Campus Drive
Cary, NC 27513
(919) 677-8000
e-mail: jbc@unx.sas.com

APPENDIX

Visual Basic Code

```
'Launch Excel and make it visible
Private Sub LAUNCHXL_Click()
  set excelobj =
  CreateObject "Excel.Application.5")
  excelobj.visible = TRUE
End Sub
```

OLE Automation with SCL

```
LAUNCHXL:
hostcl = loadclass('sashelp.fsp.hauto');
call send(hostcl, '_NEW_', excelobj, 0,
  'Excel.Application.5');
call send (excelobj, '_SET_PROPERTY_',
  'Visible', TRUE);
```

Visual Basic Code

```
'Create a new worksheet
Dim wbsobj, wobj As Object
Set wbsobj = excelobj.Workbooks
wbsobj.Add
set wsobj = excelobj.ActiveSheet
```

OLE Automation with SCL

```
call send(excelobj, '_GET_PROPERTY',
  'Workbooks', wbsobj);
call send(wbsobj, '_DO_', 'Add');
call send(excelobj, '_GET_PROPERTY_',
  'ActiveSheet', wsobj);
```

Visual Basic Code

```
'Set the value of a cell
wsobj.Cells(row+1, col).Value = var
```

OLE Automation with SCL

```
r = row+1
call send(wsobj, '_COMPUTE_', 'Cells', r, col,
  retcell);
call send(retcell, '_SET_PROPERTY_', 'Value',
  var);
```

Visual Basic Code

```
'Close Excel
Private Sub EXITXL_Click()
  excelobj.ActiveWorkbook.Close("FALSE")
  excelobj.Quit
End Sub
```

OLE Automation with SCL

```
QUITXL:
call send(excelobj, '_GET_PROPERTY_',
  'ActiveWorkBook', awbobj);
call send(awbobj, '_DO_', 'Close', 'FALSE');
call send(excelobj, '_DO_', 'Quit');
call send(excelobj, '_TERM_');
return;
```