CHAPTER

9

Using OLE in SAS/AF Software

About OLE

OLE is a means of integrating multiple sources of information from different applications into a unified document. These objects can include text, graphics, charts, sound, video clips, and much more.

OLE 1.0, which the SAS System under Windows has supported since Release 6.08, allowed you to link and embed OLE objects into SAS/AF FRAME entries and SAS/EIS applications. OLE 2.0, which Version 8 supports, provides many new features that you can use to enhance your SAS/AF frames and SAS/EIS applications.

Note: The SAS System under Windows (and OLE 2.0 in general) still supports all the features from OLE 1.0. △

Version 8 of the SAS System under Windows can function as an object container or *client*. The applications that create (and update) the objects you place in a FRAME entry are known as *servers*. You can also use the SAS System as a server from within other applications through OLE automation. For more information, see Chapter 10, "Controlling the SAS System from Another Application Using OLE," on page 209 .

For more information about OLE in general, see the documentation for the Microsoft Windows operating system. For descriptions of the error messages you might receive while using OLE features in SAS/AF software, see "Using OLE" on page 506.

SAS/AF Catalog Compatibility

SAS/AF catalogs that contain OLE HSERVICE entries can be ported from Release 6.09 for Windows NT and Release 6.10 or later for Windows transparently, just by assigning libnames to those catalogs in your Version 8 SAS session.

Note: SAS/AF catalogs created in Version 8 that contain HSERVICE entries can be ported back to Release 6.08 using the V608 option of the CPORT procedure, but the features that are ported are limited to those available in Release 6.08. △

HSERVICE entries are only usable on the host platform in which they were created. That is, any OLE features that you include in your SAS/AF applications using the SAS System under Windows cannot be ported to another host. (For portability purposes, all variations of Microsoft Windows are considered a single platform.)

Inserting an OLE Object in a FRAME Entry

SAS provides three items on the object Selection List to facilitate OLE:

OLE - Insert Object
inserts an OLE object as a new object of the type associated with a registered server application, as an object created from an existing file, or as an OLE control.

OLE - Paste Special
pastes an OLE object to the FRAME entry from the Windows clipboard.

OLE - Read Object
creates an object that references an existing HSERVICE entry in a SAS catalog.

These three items correspond to the three OLE classes in SAS/AF software: INSERT, PASTE, and READOLE.

In addition to using the Selection List to insert objects, you can select and drag objects from other Windows applications and drop them onto an open FRAME entry (in BUILD mode, or during run time if the frame or work area object is registered as a drop site for the SAS_DND_OLEOBJ representation).

Inserting an OLE Object

To insert an OLE object in a FRAME entry:

1. From the COMPONENTS window, select the **V6 objects** item to expand the object tree.
2. Scroll through the list of objects in the Selection List and select **OLE - Insert Object**.
3. Select a position for the object in the FRAME entry. As you move the mouse, the mouse pointer moves the outline of the object you want to insert. Click on the mouse button to place the object. The Insert Object dialog box appears.
4. Select the type of object you want to insert. The list of objects available to you depends on which OLE-capable applications are registered on your system. Selecting a type of object will insert an object of that type into the FRAME entry.

   Alternatively, you can create an object from a file by clicking on **Create from File**. The file you specify must have been created by one of the applications you have available to supply OLE objects. For example, if you have Microsoft Excel installed on your system, you can create an object from an Excel spreadsheet file. You also have the option of making it a linked object (instead of embedded). For more information about linked objects, see "Using Linked OLE Objects" on page 193.

   When you have selected the type of object or filename to insert, click on OK. SAS then displays the OLE - Insert Object Attributes dialog box.
5. Enter a name for the object entry in the **Entry** field. Optionally, you can also change the **Name** of the object. Two-level HSERVICE names are allowed, defaulting to the current catalog.

   Note: The HSERVICE entry is not created until you **Save** or **End** the FRAME editing session. △

   Click on OK. SAS inserts the object in the FRAME entry, displaying a representation of the object at the position you selected. If you are creating the object as new (that is, you are not creating it from an existing file), then the object automatically enters an editing session with the server. If the object server supports visual editing, then this editing session uses visual editing. For more information about visual editing, see "Editing an OLE Object within a FRAME Entry" on page 191.

Pasting an OLE Object from the Clipboard

To paste an OLE object from the Windows clipboard:

1. From another Windows application, copy or cut to the Windows clipboard the object or data you want to include in your FRAME entry.
2. From the COMPONENTS window, select the **V6 objects** item to expand the object tree.
3. Scroll through the list of objects in the Selection List and select **OLE - Paste Special**. The Paste Special dialog box appears.

4 Select the type of OLE object you would like to insert based on the clipboard contents. This is determined by the application from which you copied the data. (For example, you would typically paste Microsoft Word data as a Microsoft Word object.)

5 If you want the OLE object to link to the data instead of embed the actual data in the FRAME entry, choose **Paste Link** on the Paste Special dialog box. For more information about linked objects, see "Using Linked OLE Objects" on page 193.

Note: If you paste data from a temporary source (such as a document that you did not save), SAS will be unable to locate the data source when it attempts to link to it later when it no longer exists. You should save your data file before copying it to the Windows clipboard. △

6 After you select the type of object to paste, click on OK. SAS then displays the OLE - Paste Special Attributes dialog box.

7 Enter a name for the object entry in the **Entry** field. Optionally, you can also change the **Name** of the object. Two-level HSERVICE names are allowed, defaulting to the current catalog. Note that the HSERVICE entry is not created until you **Save** or **End** the FRAME editing session.

Click on OK. SAS pastes the object in the FRAME entry, displaying a representation of the object at the position you selected.

Reading an OLE Object from an HSERVICE Entry

To read an existing OLE object stored as an HSERVICE entry in a SAS catalog:

1 From the COMPONENTS window, select the **V6 objects** item to expand the object tree.

2 Scroll through the list of objects in the Selection List and select **OLE - Read Object**. SAS displays the OLE - Read Object Attributes dialog box.

3 Enter the name of the HSERVICE entry in the **Entry** field. Two-level HSERVICE names are allowed, defaulting to the current catalog. To use the Select window to find the entry, click on the arrow next to the **Entry** field.

Click on OK. SAS inserts the object in the FRAME entry, displaying a representation of the object at the position you selected.

Note: You cannot change the name of an HSERVICE entry that you read in. If you want to assign a different name to the HSERVICE entry, copy the HSERVICE entry to a new name before you read the object. △

Inserting an OLE Object Using Drag and Drop

To insert an OLE object into a FRAME entry by dragging and dropping it:

1 Create the object using the server application. For example, if you want to embed a Microsoft Excel chart object into your FRAME entry, use Microsoft Excel to create the object. Or, you can select an OLE object that is embedded in another application.

2 With both SAS and the server application running, arrange the application windows so that both the server application (with the object) and the SAS BUILD: DISPLAY window (with the FRAME entry) are visible on the screen.

3 Select the object in the server application. With the mouse button depressed, drag the object from the server application to the position in the FRAME entry where you want to place the object. The cursor changes to a box with an arrow,

indicating that the FRAME entry is a valid place to drop the object. Note that you do not need to draw a region in the FRAME to insert the object. You can also use drag modifier keys, as discussed in "Changing the Action of the Drag and Drop" on page 191 to control the drag and drop behavior.

When you release the mouse button ("dropping" the object), SAS inserts the object into the FRAME, automatically creating a name and an HSERVICE entry for the OLE object. SAS displays a representation of the object at the position you selected.

Dragging and Dropping OLE Objects During Run Time

You can allow the dragging and dropping of OLE objects while your SAS/AF application is running. To enable this, you must register the OLE object type with a valid drag and drop representation.

OLE objects must be registered with the SAS_DND_OLEOBJ representation. For more information about registering objects for drag and drop, see *SAS/AF Software: FRAME Application Development Concepts* and the Widget Class in *SAS/AF Software: FRAME Class Dictionary*.

Changing the Action of the Drag and Drop

By default, dragging an OLE object from another application into SAS moves the object (unless the object is of a type that can only be read and not removed). You can override this default action by using a *drag modifier*; that is, a key press that indicates that you want to perform a different drop action:

- To copy an object from the server application, hold down the Ctrl key when you drop the object on the target window. When you press the Ctrl key, the cursor changes to an arrow with a box and a plus (+) sign.
- To create a link to the data in a SAS/AF FRAME entry, hold down the Ctrl and Shift keys when you drop the object on the BUILD window. When you press the Ctrl and Shift keys, the cursor changes to an arrow with a box and an equal (=) sign. (This feature might vary based on the other application.) Remember not to paste a linked object from a temporary source, as SAS cannot locate a data source when it no longer exists.

Alternatively, you can initiate a *nondefault* drag and drop action (if the server application supports it). Use the *right* mouse button to select the object and drag and drop it into the FRAME entry. When you release the mouse button, SAS displays a pop-up menu allowing you to select whether to move, copy, or link to the object. The choices in the pop-up menu might vary among different types of OLE objects.

Editing an OLE Object within a FRAME Entry

One of the most impressive features of OLE 2.0 is visual editing–the ability to edit an embedded object in-place, without explicitly changing to another application.

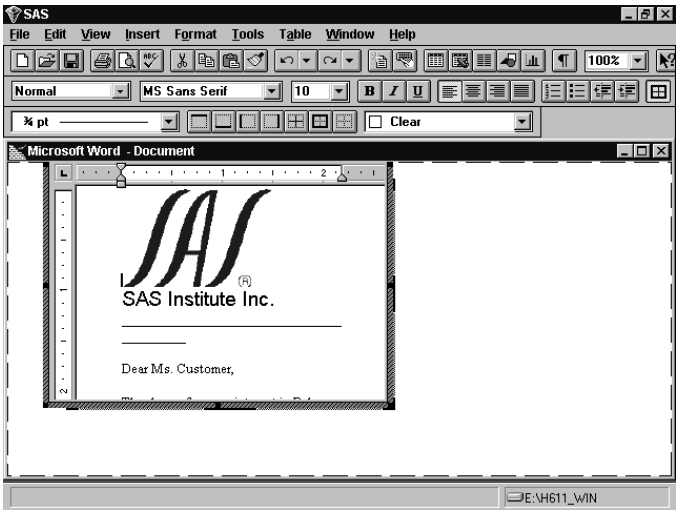
To activate visual editing for an OLE object in your FRAME entry at build time, click the right mouse button and select Edit. To activate visual editing at run time, simply double-click on the object. If the object's application supports visual editing as a server application, then the following occurs:

- The object's representation in the FRAME entry changes to an editing session of the actual object. The object's borders might change to accommodate the tools supplied by the server application.

- The SAS menu bar changes to accommodate the menu bar of the server application. The **File** and **Window** menu remains the same, but the remainder of the menu bar changes to that of the server application.
- If the server application normally provides any tools, such as toolbar icons or a floating toolbox, those items also become available.

For example, Display 9.1 on page 192 shows a SAS/AF FRAME entry with a Microsoft Word object activated.

Display 9.1 SAS/AF FRAME Entry with Word Object Activated



After this transformation, you can edit the object using all of the tools and menus provided by the server application.

To end your visual editing session, click elsewhere inside the FRAME entry and outside the object. SAS resumes control of the session, and returns to the default SAS menus and tools.

Note:

1. The HSERVICE entry is automatically updated at the end of a visual editing session *only* if the object has been saved previously (that is, an HSERVICE entry has been created for it). Otherwise, you must select **Save** (or **End**) from the **File** menu in SAS/AF software to create the HSERVICE entry.

   Also, if you modify the object during TESTAF mode and you want to save the modifications in the HSERVICE entry, you must update the object's contents by selecting **Update** from the **Locals** menu before returning to BUILD mode.
2. If you move the OLE object within the FRAME entry during visual editing (in BUILD mode), the object returns to its original position when you click outside of it (ending the visual editing session). If you want to move the object to another position in the FRAME entry, end the visual editing session and then move the object region.
3. Most OLE objects require that you double-click on them to activate them. However, a few types of objects require only a single-click to activate them.
4. If you attempt to edit a linked object or an OLE object whose server application does not support visual editing, the server application launches as a separate instance and allows you to edit the object. This is known as *open editing* and is consistent with the behavior of linked objects and all OLE 1.0 objects.

△

Invoking OLE Verbs

Each OLE object (except OLE controls) has a default action that it performs when you double-click on it. For many objects, the default action is **Edit** (invoking a visual editing session for OLE 2.0 or an open editing session for linked objects and all OLE 1.0 objects). However, there are some objects for which the **Edit** action is secondary (for example, a Media Clip object, where **Play** is the primary action). Also, many objects have more than one action that they can perform, so they understand more than one OLE verb. (Note that double-clicking on an OLE object in BUILD: DISPLAY mode does not perform the default action, but double-clicking on the object in TESTAF mode does.)

To access the menu of OLE verbs for an OLE object in BUILD mode, click on the object with the *right* mouse button. The name of the OLE object is located at the bottom of the pop-up menu. In the cascading menu off that item, there is a list of valid OLE verbs for the object. Select a verb from this menu to perform that action. The default verb appears first in the list of verbs.

For example, a Microsoft Excel object understands **Edit** (for visual editing) and **Open** (for open editing). A Media Clip object understands **Play** and **Edit**.

You can also access the list of valid verbs by clicking on the **Associated Verbs...** item in the Object Attributes dialog box for the object. This list just contains the names of the verbs; you cannot initiate the verbs from here. Again, the verb at the top of the list is the default verb.

Using SCL, you can invoke any verb that a particular OLE object understands by using the _EXECUTE_ method with the verb as an argument. For example, this code would invoke the verb **Play** on the OLE object **mediaobj**:

```
call notify('mediaobj', '_EXECUTE_', 'Play');
```

You can specify multiple verbs in a single call to _EXECUTE_. For more information about the _EXECUTE_ method, see "_EXECUTE_" on page 496.

Using Linked OLE Objects

A linked OLE object contains information about the object's server application and points to the data file that resides on disk, but does not contain data for the object itself. The object contains a static picture that represents the contents of the linked source.

Using the Links dialog box, you can specify to update a linked object:

- automatically, whenever you update the source file that the object points to. (You must reload the FRAME entry before it reflects the change.)
- manually, by choosing Update Now in the Links dialog box or by using the _UPDATE_ method in SCL.
- manually, by pointing the object to a different source file using either the Links dialog box or the _UPDATE_ method in SCL.

Linked OLE objects that you include in a FRAME entry:

- support open editing only (as opposed to visual editing, described in "Editing an OLE Object within a FRAME Entry" on page 191). When you double-click on the object's representation in the FRAME, the server application is invoked in a separate window with the object's data file open.

  You can also update the data of a linked object by using the server application to open the data file the object points to.
- must point to existing data files. If you change the location of a data file to which an object is linked, you must update the links information for the object.

If you create a linked object using **OLE - Paste Special**, the data source that you paste from must be permanent (that is, you must have saved it to disk). If you create a linked object from a temporary data source, SAS will be unable to locate the data to update the object when the data source no longer exists.

Updating a Linked Object with the Links Dialog Box

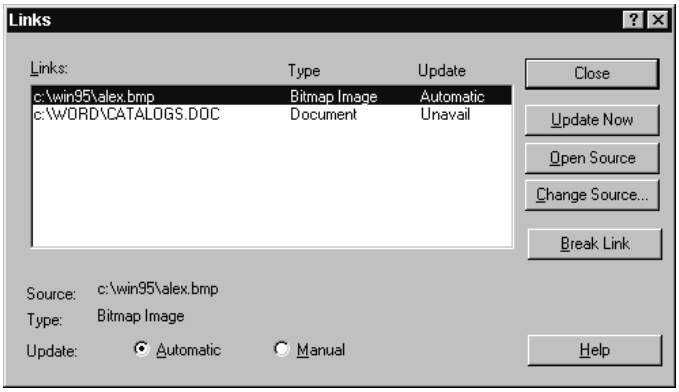
To update the links information with the Links dialog box (shown in Display 9.2 on page 194):

1. Click on the object with the right mouse button. A pop-up menu appears, with the object type listed as the bottom menu item.
2. Click on the bottom menu item. A cascading menu containing valid OLE verbs for the object appears.
3. Click on **Links...**. The Links dialog box appears, containing link information for all of the linked objects in the FRAME entry. (If there are no linked objects in the FRAME, then the **Links...** item is disabled.)
4. Use the Links dialog box to change information about the object as necessary. For example, if the data file resides in a different location, you can change the source for the object link.

An alternate way to open the Links dialog box for a linked OLE object is to use the DLGLINKS command from the command line. You can also use the _EXECUTE_ method in SCL to invoke the DLGLINKS command. For example:

```
call notify('linkobj','_execute_','dlglinks');
```

Display 9.2 Links Dialog Box



Updating a Linked Object Programmatically

To change the source of a linked object programmatically with SCL, use the _UPDATE_ method to specify a new HSERVICE entry to associate with the object. The _UPDATE_ method for OLE objects accepts the name of an HSERVICE entry as a third argument. (This method overrides the Widget class _UPDATE_ method.) For the syntax of the OLE _UPDATE_ method, see "_UPDATE_" on page 500.

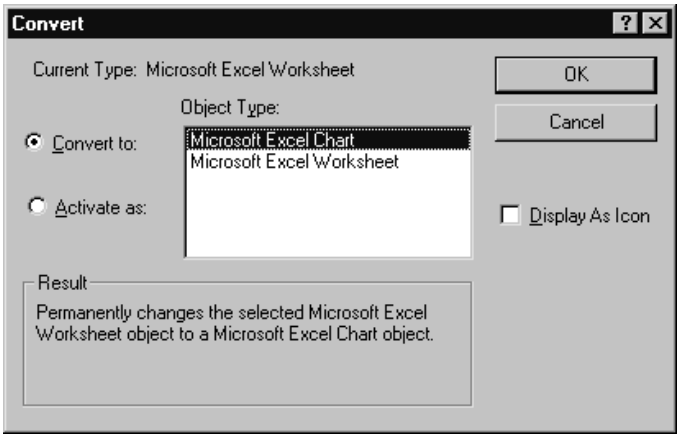
Converting OLE Objects

An OLE object can be associated with only one server application, but some OLE objects can be converted for use with a different server application than the one that created them.

You can convert an object by using the Convert dialog box (shown in Display 9.3 on page 195). This dialog box lets you:

- change the object's view from an icon to object content and vice-versa.
- change the object's type from one server application to another. For example, you can convert a Microsoft Excel object to a Lotus 1-2-3 object, provided that you have the server application that can convert the object on your system. This type of conversion is permanent.
- activate the object with a different server application than originally created it, without altering the object type. For example, you can choose to activate a Lotus 1-2-3 object using Microsoft Excel as a server. This allows you to edit the object as if it were an Excel object. The object continues to be a Lotus 1-2-3 object. All subsequent Lotus 1-2-3 OLE objects you create will use Excel as an OLE server, unless you change the conversion settings again.

Display 9.3 Convert Dialog Box



To convert an OLE object within a SAS/AF FRAME entry:

1. Click on the object with the *right* mouse button.
2. At the bottom of the pop-up menu, select the object's name (thus revealing the cascading menu).
3. In the cascading menu, select **`Convert...`**. The Convert dialog box appears, listing the valid object types to which you can convert the selected object.
4. If you want to actually convert the object to another type, select the desired target object type and click on OK.

   If you want to toggle the object between icon view and content view, check **`Display As Icon`**.

   If you want to activate the object using another server, click on **`Activate as`** and then select the server application to use.
5. Click on OK.

An alternate way to open the Convert dialog box for an OLE object is to select the object and issue the DLGCONVERT command on the command line. Also, you can use the _EXECUTE_ method in SCL to invoke the DLGCONVERT command. For example:

```
call notify('sheetobj','_execute_',
            'dlgconvert');
```

Automating OLE Objects and Applications

Some Windows applications provide a scripting language that allows you to control and update objects and external applications through automation. In SAS/AF software, you can use SAS Component Language (SCL) for OLE automation. Using SCL code to send instructions to the OLE object, you can update the object's data based on a user's actions in your SAS/AF application.

In SAS/AF software, you can automate:

- OLE objects embedded in a FRAME entry, using the OLE class
- OLE objects linked to a FRAME entry, using the OLE class
- OLE applications not associated with a FRAME entry, using the OLE Automation class.

Using SCL, you can communicate with any OLE object or application that supports OLE automation as a server. In this communication, SAS acts as a client while the automation application acts as a server. The server provides OLE automation objects, which you can control with SCL code. Using SCL methods, you can send OLE methods to the server for execution. You can also get and set the properties of the objects you control. OLE automation servers can support multiple types of objects, each of which can have a unique set of methods and properties. The SCL methods you can use are listed in Table 9.1 on page 196 and described in detail in "Summary of OLE Class Methods" on page 493.

Note: Do not confuse the SCL OLE automation methods (listed in the table) with the methods provided by the OLE automation server. In SAS/AF software, the _COMPUTE_ and _DO_ SCL methods provide access to the methods supported by the OLE automation server. Each OLE automation server supports different methods, but you must always use the _COMPUTE_ or _DO_ method in SCL to invoke them. (You can use subclassing to create new methods that encapsulate these. For an example, see .) △

Table 9.1 OLE Automation Class Methods

OLE Automation Method	Description
COMPUTE	invokes a method supported by the OLE automation server and returns a value
_DO_	invokes a method supported by the OLE automation server (with no return value)
_GET_PROPERTY_	retrieves the value of a property exposed by the OLE automation server
_GET_REFERENCE_ID_	returns the reference identifier of an object provided by the OLE automation server
_IN_ERROR_	returns an object's ERROR status

OLE Automation Method	Description
NEW	assigns an SCL identifier to an external instance of an OLE automation server
_SET_PROPERTY_	sets the value of a property exposed by the OLE automation server

Note: The return values and arguments passed between the automation server and SAS using the OLE automation methods are passed by value, not by reference–including those arguments that the server defines as pass-by-reference. That is, the arguments contain actual static values, not pointers to values that you can modify. △

Accessing Array Values Returned by the OLE Automation Server

The SAS System lets you access single-dimensional arrays that are passed back by the OLE automation server as a property or as a result of one of its methods. When the array is returned to SAS, SAS stores it in an SCL list.

For example, the following SCL code creates and populates a listbox in a Microsoft Excel worksheet and stores the contents of the listbox in an SCL list:

```
list=makelist();  /* create the SCL list */
   /* Add a Listbox in a worksheet */
call send(worksht, '_COMPUTE_', 'Listboxes',
          listbox);
call send(listbox, '_DO_', 'Add', 20, 50,
          40, 100);
call send(worksht, '_COMPUTE_', 'Listboxes',
          1, listone);
   /* Fill the Listbox with a range of */
   /* values from the worksheet        */
call send(listone, '_SET_PROPERTY_',
          'ListFillRange', 'A1:A3');
   /* Get the contents of the Listbox */
call send(listone, '_GET_PROPERTY_',
          'List', list);
```

Note: The SAS System does not support passing arrays (or SCL lists) as arguments to an OLE automation server; it supports only receiving the array values returned from the server as a result of the _GET_PROPERTY_ or _COMPUTE_ methods. Also, SAS does not support multidimensional arrays as values supplied to or returned by an OLE automation server. △

Using Value Properties

OLE automation servers (including OLE custom controls) can designate one of their properties or methods as a *value property*, which is used as the default property or method when the automation code accesses an object provided by the server without explicitly specifying a property or method name.

In SCL, you can access the value property of a server by specifying an empty string in place of the property name when invoking _GET_PROPERTY_ or _SET_PROPERTY_, or in place of the method name when using _DO_ or _COMPUTE_. For example, if the Text property is the value property, then the following code:

```
call notify('sascombo', '_set_property_', '',
            'An excellent choice');
```

is equivalent to:

```
call notify ('sascombo', '_set_property_',
             'Text', 'An excellent choice');
```

Both the SAS ComboBox and SAS Edit controls (supplied with the SAS System) designate Text as their value property.

Specifying Optional Parameters in OLE Server Methods

Some OLE server applications expose methods that have optional parameters; that is, if you do not specify a value for one or more of the parameters that a method supports, the OLE server uses a default value for those parameters. Refer to the documentation for the OLE server application you are using for information about which parameters are optional.

The SAS System supports the use of optional parameters by letting you specify a SAS missing value in place of the parameter you want to omit. The default missing value character is a period (but that can be changed by using the MISSING system option).

For example, Microsoft Excel supports a ChartWizard method that accepts 11 arguments, most of which are optional. This SCL code invokes this method with all of its arguments:

```
call send(chart, '_DO_', 'ChartWizard', hcell,
                 -4098, 6, 1, 0, 0, 1,
                 "Automation at work!",
                 'Column', 'Value', 'Row');
```

Here is the equivalent SCL code that omits the optional parameters (substituting the missing value character):

```
call send(chart, '_DO_', 'ChartWizard', hcell,
                .,.,.,.,.,.,
                "Automation at work!",
                 .,.,.);
```

Note: Your SCL code must still respect the position of the optional parameters when invoking methods. When you specify a missing value character as an argument, it must be in place of a parameter that is optional to the OLE server's method. △

Creating an External OLE Automation Instance

External OLE Automation Instances can be for an application on your local machine or an application on a remote machine. Before you can automate an external OLE application, you must create an instance of the OLE Automation class. (Note that this is not necessary when you automate objects that you embed or link in your FRAME entry, because placing them in the FRAME entry creates the instance for you.) Unlike the OLE class, the OLE Automation class is not derived from the Widget class and, therefore, has no visual component to include in a FRAME entry. Instead, you must load an instance of the HAUTO class (using the LOADCLASS function) in the SCL code that drives the automation. For example:

```
hostcl=loadclass('sashelp.fsp.hauto');
```

After you create an instance of the OLE Automation class, you must associate the new instance with an SCL object identifier (which you need to use when calling

methods with CALL SEND) and an OLE server application. To obtain the identifier, use the _NEW_ method on the newly created instance of the OLE Automation class. This example stores the object identifier in **oleauto** and associates the object with Microsoft Excel (which has the identifier **Excel.Application.8** in the Windows registry) on the local machine.

```
call send(hostcl, '_NEW_', oleauto, 0,
          'Excel.Application.8');
```

To create an instance of the OLE Automation class for a remote machine, the remote machine must be configured to permit the user to start remote instances using Distributed COM Configuration Properties (DCOMCNFG.EXE). In Windows NT, DCOMDNFG.EXE is located in the \WINNT\SYSTEM32 folder. In Windows 95 and Windows 98, DCOMDNFG.EXE is located in the \WIN9x\SYSTEM folder. For more information on Distributed COM Configuration Properties, see your Windows documentation. The following example creates an instance of Microsoft Excel on a remote machine. Once created, the method and property calls to that instance work as if it were on a local machine.

```
Init:
  HostClass = loadclass('sashelp.fsp.hauto');
  ExcelObj = 0;

  /* Define the machine name and put it in a list  */

  machineName = '\\Aladdin';
  inslist = makelist();
  attrlist = makelist ();

  rc = insertc (attrlist, machineName, -1, 'remoteServer');
  rc = insertl (inslist, attrlist, -1, '_ATTRS_');

  /* Instantiate the Excel object and make it visible */

  call send (HostClass, '_NEW_',ExcelObj, inslist,
  'Excel.Application');
  call send (ExcelObj, '_SET_PROPERTY_', 'Visible', -1);
return;
```

For more information about the _NEW_ method, see "_NEW_" on page 499.

After you create an instance of an OLE Automation object, you can automate that object in much the same way you would automate an object that you have embedded or linked in your frame. The following table notes some key differences between the types of objects.

SAS OLE objects...	SAS OLE Automation objects...
are derived from the Widget class.	are derived from the Object class.
have a visual component (the object you place in the FRAME entry).	have no visual component within the FRAME entry.
are created by placing the object in a region in the FRAME entry (using drag and drop).	are created by using the LOADCLASS statement and the _NEW_ method in SCL.

SAS OLE objects...	SAS OLE Automation objects...
represent the specific type of data object (which you choose) supported by the OLE server.	represent the top-level application object supported by the OLE server, which you then might use to open objects of specific data types.
allow you to call methods with CALL NOTIFY by passing in the object name from the FRAME entry.	require you to call methods with CALL SEND, passing in the object identifier returned by the _NEW_, _GET_PROPERTY_, or _COMPUTE_ methods.

Example: Populating a Microsoft Excel Spreadsheet with SAS Data

Table 9.2 on page 200 contains SCL code to populate a Microsoft Excel spreadsheet with data from a SAS data set.

Table 9.2 SCL Code for Populating a Microsoft Excel Spreadsheet

Load an instance of the OLE Automation class and invoke Excel. Set the object to Visible so you can see the automation in progress.	`LAUNCHXL:`
	
	`hostcl = loadclass('sashelp.fsp.hauto');`
	`call send(hostcl, '_NEW_', excelobj, 0,`
	`'Excel.Application');`
	`call send(excelobj, '_SET_PROPERTY_', 'Visible', 'True');`
	`return;`

Get the identifier for the current Workbooks property and add a worksheet. Then get the identifier for the new worksheet.	`CREATEWS:`
	`call send(excelobj, '_GET_PROPERTY_', 'Workbooks',`
	`wbsobj);`
	`call send(wbsobj, '_DO_', 'Add' );`
	`call send(excelobj, '_GET_PROPERTY_', 'ActiveSheet',`
	`wsobj);`

Open a SAS data set.	`dsid=open('sasuser.class','i');`
	
	`call set(dsid);`
	`rc=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,'_SET_PROPERTY_', '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);`
	`  call send(retcell, '_SET_PROPERTY_', 'Value' ,var);`
	`  end;`
	`  rc=fetch(dsid);`
	`  end;`
	`end;`
	`dsid=close(dsid);`
	`return;`

Close the worksheet and end the Excel session. The _TERM_ method deletes the OLE automation instance.	`QUITXL:`
	`call send(excelobj,'_GET_PROPERTY_', '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, 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.

Table 9.3 on page 201 shows some excerpts of Visual Basic code and their SCL equivalents.

Table 9.3 Visual Basic Code Samples and Their SCL Equivalents

Visual Basic Code	OLE Automation in SCL

Launch Excel and make it visible
`Set excelobj = CreateObject("Excel.Application")`
`excelobj.Visible = True`
	`hostcl = loadclass('sashelp.fsp.hauto');`
	
	`call send ( hostcl,  '_NEW_', excelobj, 0,`
	`  'Excel.Application');`
	`call send (excelobj,'_SET_PROPERTY_',`
	`  'Visible','True');`

Create a new worksheet
`Dim wbsobj, wsobj As Object`
`Set wbsobj = excelobj.Workbooks`
`wbsobj.Add`
`Set wsobj = excelobj.ActiveSheet`
	`call send(excelobj,'_GET_PROPERTY_',`
	`  'Workbooks', wbsobj);`
	`call send(wbsobj,  '_DO_', 'Add');`
	`call send(excelobj,'_GET_PROPERTY_',`
	`  'ActiveSheet', wsobj );`

Visual Basic Code	OLE Automation in SCL
Set the value of a cell	

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

Close the Excel application object	

`excelobj.ActiveWorkbook.Close ("False")`
`excelobj.Quit`
	`call send(excelobj,'_GET_PROPERTY_', 'ActiveWorkbook', awbobj);`
	`call send(awbobj, '_DO_', 'Close', 'False');`
	`call send(excelobj,'_DO_', 'Quit');`
	`call send(excelobj,'_TERM_');`

Using OLE Custom Controls (OCXs) in Your SAS/AF Application

An OLE custom control is a special type of OLE object or collection of OLE objects that has an interface to expose its own properties and methods. You can control these objects through its graphical interface and with SCL code.

OLE custom controls differ from other OLE objects in these ways:

- They generate events based on user actions, which you can respond to in your FRAME entry. Note that the object's SCL label is not run by default when you activate an OLE control.
- They assume ambient properties (such as color and font) based on the environment in which they are used.

OLE controls are packaged in their own dynamic linked library (with a file extension of OCX). Using SCL code, your FRAME entry can respond to events generated by the OLE control (mouse clicks, key presses, and so on). The events exposed by OLE controls vary among controls. For a list of events, see the documentation for the control you are using. After inserting the control into the FRAME entry, you can view the event map by selecting **Object Attributes** for the OLE control object and then **Event Map**.

Note: The OLE controls that SAS provides require 32-bit containers, which makes them unusable with Windows applications that offer only 16-bit container support. Also, because SAS is a 32-bit container, you cannot use 16-bit controls with it. △

Inserting an OLE Control in a FRAME Entry

To insert an OLE control in a FRAME entry:

1. From the COMPONENTS window, select the **V6 objects** item to expand the object tree.
2. Double-click on **OLE - Insert Object** from the Selection List. The Insert Object dialog box opens. You can also drag **OLE - Insert Object** from the Selection List to the BUILD window. When you release the mouse button the Insert Object dialog box opens.
3. Select the **Create Control** radio button to display a list of registered OLE custom controls. If the OLE control you want to use is not listed here and you have it on your system, you need to register the control (see "Registering OLE Controls" on page 203).
4. Select the name of the OCX control you want to insert.

Registering OLE Controls

Before you can use any OLE control in Windows, the control must be registered with Windows. SAS ComboBox and SAS Edit, the two OLE controls provided with the SAS System, are automatically registered when you install the SAS System.

If you want to install other controls for use with SAS or other applications, you must register the control with Windows (unless the control was installed by a process that performed the registration for you). The OLE control will not be available from the Insert Object dialog box until it is registered.

To register an OLE control:

1. Complete steps 1-4 as described in "Inserting an OLE Control in a FRAME Entry" on page 202 to invoke the Insert Object dialog box with the list of registered controls.
2. Click on Add Control... to invoke the Browse file selection dialog box.
3. Use the dialog box to select the control (which usually has a file extension of OCX) that you want to register.

   When you click on OK, the control is added to the list of registered controls in the Insert Object dialog box.

Accessing OLE Control Properties

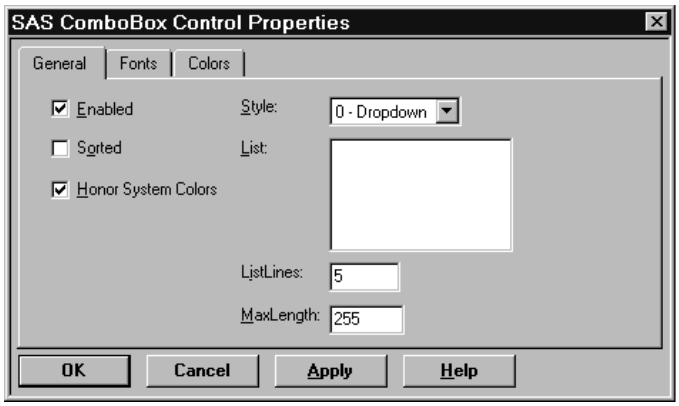
OLE controls have properties that you can set or retrieve using SCL methods. Some controls make some of their properties available through a properties page, which lets you set or retrieve the data interactively.

Accessing the OLE Control Properties Page

To invoke the properties page for a control, click on the right mouse button within the control's region in the BUILD: DISPLAY window and then select **`Properties`** from the pop-up menu. The properties page for the control appears. Display 9.4 on page 203 shows an example of a properties page for an OLE control.

OLE controls provide a Properties verb, which you can use with the _EXECUTE_ method in SCL to bring up the Properties page for the control. Or, you can access the pop-up menu for the control, then choose the cascading menu with the control's name. The Properties verb is available off that cascading menu.

Display 9.4 An Example Properties Page



You can use the properties page to view or change settings for some of the exposed properties.

Note that the control is not active (that is, you cannot interact with its interface) while you are in DISPLAY mode. The control becomes active in TESTAF mode.

Accessing Properties Using SCL Code

When you use OLE controls in a SAS/AF application, you may want to access the properties of the control programmatically. Also, an OLE control might not expose all of its properties in a properties page. You can access the properties of a control by using the _SET_PROPERTY_ and _GET_PROPERTY_ methods.

Before you can access a property, you must know:

- the object label of the OLE control in your SAS/AF FRAME entry
- the name of the property you want to access
- the type of data that the property holds.

For example, suppose you have a combo box control named **sascombo** in your FRAME entry, and you want to set the list style to **simple** (represented by the integer 1):

```
call notify ('sascombo', '_set_property_',
             'Style', 1);
```

If you want to retrieve data from a property, you must use a variable that is of the same type as the data you want to read. For example, if you want to learn what text the user specified in the edit portion of a combo box, include the following code:

```
length text $ 200;
call notify ('sascombo', '_get_property_',
             'Text', text);
```

Interacting with the OLE Control Using SCL Methods

OLE controls support methods that control their content and behavior. You use either the _DO_ or _COMPUTE_ SCL methods to send a message to an OLE control telling it to implement one of its methods.

- Use the _DO_ method in SCL when the OLE control method performs some action but does not return a value. For example, the SAS ComboBox OLE control has a method that clears all items from the list:

  ```
  call notify('sascombo', '_DO_', 'Clear');
  ```

- Use the _COMPUTE_ method in SCL when the OLE control method returns a value. You specify a variable in the SCL code that will contain the return value when the method ends. For example, the SAS ComboBox OLE control has a method that returns an item at a specified position in the list:

  ```
  length item $ 80;
  call notify('sascombo', '_COMPUTE_',
              'GetItem', 2, item);
  ```

  When this call returns, *item* contains the text of the item at position 2 (the third item in the list).

Responding to OLE Control Events

OLE controls generate events that you can respond to in your SCL code. You can create a label in your SCL code for OLE events just like you do for SAS/AF events.

Assigning SCL Code to an OLE Control Event

To assign SCL code to run when an OLE control event occurs:

1. Select the OLE control object in the BUILD window.
2. Select **`Object Attributes`** from the pop-up menu for the object.
3. Select **`Event Map`** from the Object Attributes dialog box. The Event Map dialog box appears (shown in Display 9.5 on page 205).
4. In the Event Map dialog box, select the event that you want to respond to using SCL code.
5. Specify the SCL, FRAME, or PROGRAM source entry and (if applicable) the SCL label where the event-handling code resides.

   Note: You can specify the same SCL source entry that is stored with the FRAME entry; however, in addition to compiling the code with the FRAME entry, you must also compile the SCL entry outside of the FRAME context (that is, outside of the BUILD: SOURCE and BUILD: DISPLAY windows) in order for the event handler to recognize the SCL label. Thus, it is more efficient to store event-handling code for OLE controls in an SCL source entry that is not associated with a FRAME entry. △

Display 9.5 Event Map Dialog Box

Note: Many OLE controls include a LostFocus event, which they generate when the control loses window focus. Because of the way that SAS/AF software communicates with the control, mapping the LostFocus event sometimes has the effect of placing focus back on the control that just lost it. Although you can still respond to the LostFocus event in your FRAME entry, be aware that this might cause unusual focus behavior. △

Retrieving Argument Values from Events

Some OLE control events also include parameters you might find useful. For example, the SAS ComboBox control generates a KeyPress event that also reports the ASCII value of the key that was pressed. If a particular event passes an argument back to the FRAME entry, the type of value returned is indicated in the Event Map dialog box. A numeric value is indicated with an **`N`**; a character value is indicated with a **`C`**.

To retrieve the value returned by an OLE control event, you must define a method (using the METHOD statement in SCL) in the event-handling code. In the argument list for the METHOD statement, specify a variable of the type that you expect the OLE control to return. This variable contains the value returned by the event. You can then use that variable as you wish inside your event-handler.

For example, suppose you want to retrieve the value of the key that triggered the KeyPress event in the SAS ComboBox control and then report it as an ASCII character. The KeyPress event returns an integer that represents the ASCII value of the key pressed. Your event-handling code would look like the following:

```
   /* Label specified in Event Map dialog box */
KEYPRESS:
   /* Define a method with an
      integer argument */
method keyval 8;
      /* Convert the integer to an
         ASCII character */
   keychar=byte(keyval);
  put keychar=; /* Output the character */
endmethod;
```

Example: Mapping OLE Control Events to SCL Code

When mapping OLE control events, you can do one of the following:

- Map each event in the Event Map window to a different labeled section of SCL code, with each piece of code performing different actions.
- Map all of the events in the Event Map window to a single labeled section of SCL code, use the _GET_EVENT_ method to detect which event was triggered, and act accordingly.
- Use a combination of these strategies by assigning one event (such as the Click event) to SCL code that runs the object's label (using the _OBJECT_LABEL_ method), and map the remaining events to a single label that uses the _GET_EVENT_ method to determine the event and appropriate action. The object's label is not run by default for OLE controls.

The following example shows how to structure the SCL code when all events for an OLE control are mapped to a single label, which in turn runs the object's label to determine the event and act accordingly:

```
length event $ 80;
  /* All OLE control events are mapped to
     this label */
RUNLABEL:
     /* Call the object's label */
  call send(_self_, '_OBJECT_LABEL_');
return;
  /* This is the label of the OLE control */
OBJ1:
     /* Determine the last event */
  call notify('obj1', '_GET_EVENT_', event);
  select (event);
     when('Click') put 'Click received';
     when('DblClick') put 'DblClick received';
     otherwise put event=;
  end;
return;
```

Example: Subclassing an OLE Custom Control

If you create SAS/AF applications that make frequent use of one or more OLE custom controls, you might want to write your own methods to abstract the methods

that the control recognizes without having to specify the intermediate _DO_ and _COMPUTE_ methods in SCL.

You can achieve this by creating a subclass of the OLE class and adding methods to your derived class. When you insert the OLE control into your FRAME entry, be sure to insert it as an instance of the new class that you define (instead of **OLE - Insert Object**). The examples provided here contain sample code you can use to abstract the methods of a control. They do not include details about how to create subclasses. For information about creating subclasses of a SAS/AF classes, see *SAS/AF Software: FRAME Class Dictionary*.

Adding an Item to a Combo Box List

You can use this method to add a new item to the list portion of the SAS ComboBox control. The SAS ComboBox control uses zero-based numbering to indicate the positions of the list items (that is, the first item is at position 0, the second is at position 1, and so on). The following method lets you specify the position numbers such that position 1 holds the first item.

```
/* Add a new item to a ComboBox list.  */
ADDITEM:
method text $200 row 8 rc 8;
     /* adjust for zero-based index */
  ocxrow = row-1;
  call send(_self_, '_COMPUTE_', 'AddItem',
            text, ocxrow, rc);
  if ( rc = 0 ) then
    _MSG_="ERROR: Could not add item to list.";
endmethod;
```

Assuming you mapped this code to a new method called ADD_ITEM, you would use this syntax to add a new item to the control:

```
   /* Adds 'Item 1' at the first position */
   /* in the control                      */
length success 8;
call notify('sascombo', 'ADD_ITEM',
            'Item 1', 1, success);
```

Finding an Item in a Combo Box

The following method finds the specified item and returns its position in the list. As in the previous example, this method adjusts the position number to be one-based instead of zero-based.

```
FINDITEM:
method text $200 row 8;
  call send(_self_, '_COMPUTE_', 'FindItem',
            text, row);
  row = row + 1; /* adjust for zero-based */
endmethod;       /* index                */
```

Assuming you mapped this code to the FIND_ITEM method, you would then use it as in this example:

```
length position 8;
call notify('sascombo','FIND_ITEM',
            'Lost Item', position);
```

Retrieving the Text Value of the Control

Both the SAS ComboBox and SAS Edit controls have Text properties, which you can access using the _GET_PROPERTY_ method with the property name. For easier and more intuitive access from your OLE subclass, you can override the _GET_TEXT_ method and map it to this code:

```
GETTEXT:
method text $200;
   call send(_self_, '_GET_PROPERTY_',
             'Text', text);
endmethod;
```

You would then access the Text property of a control the same way you access the text of other SAS/AF widget objects:

```
length text $ 200;
call notify('sasedit', '_GET_TEXT_', text);
```

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Companion for the Microsoft Windows Environment, Version 8,* Cary, NC: SAS Institute Inc., 1999. pp.555.

SAS Companion for the Microsoft Windows Environment, Version 8

ISBN 1–58025–524–8

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, September 1999

The Institute is a private company devoted to the support and further development of its software and related services.