

OLE: The SAS System for Windows Enters the Bull Ring

Robin E. Way
Northwest Natural Gas
Portland, Oregon

Abstract

Object Linking and Embedding (OLE) is a major component of inter-application collaboration on the Microsoft Windows family of platforms. The OLE framework is used to integrate various types of information (such as text, graphics, sound, and video) by representing each piece of information as "objects." The SAS System for Windows (version 6.11, Wave 2) now supports three types of OLE functionality:

- containers (to assemble objects from other applications)
- automation (to control the properties and actions of other applications)
- controls (to make 3rd party application development tools available to other applications)

This paper clarifies the usefulness of these OLE functions, provides some practical examples of how OLE can be used today with the SAS System and other business software, and demonstrates how progress in OLE-related applications development will make the future boundaries between the SAS System and other applications a little more transparent.

Overview of Windows and OLE

The popularity of the Windows family of platforms has provided a healthy growth market for SAS Institute. According to the SAS Institute 1995 Annual Report, PC platform-related revenue growth was 40% over the previous year; nearly 70% of all new sales are in the PC platform; and internationally the PC platforms numbered over 15,500 installed sites. In the Pacific Northwest, installed sites running on the various flavors of Windows account for 68% of all PC platform sites, and 27% across all platforms.

SAS Institute has performed well in expanding the functionality of their Windows products to meet the conventions and standards set by Microsoft and other desktop product vendors. Release 6.11 (Wave 2) of the SAS System for Windows meets Microsoft's "Designed for Windows 95" logo standards.

Most importantly, from the perspective of SAS users and applications developers, SAS Institute has pursued research and development to take advantage of evolving application connectivity tools, such as ODBC, DDE, SQL, and OLE, the topic of this presentation.

To understand SAS Institute's rationale for pursuing OLE as an application connectivity framework, we need to understand some of the principles underlying the more recent development of Microsoft Windows family of platforms. This family includes (in chronological order) Windows 3.1, Windows 3.11 for Workgroups, Windows NT 3.51, Windows 95, and the upcoming Windows NT 4.0, now in its second beta release. There are two important elements of Windows development that are particularly relevant to a discussion of OLE: the Windows object model concept, and the components of the OLE framework.

Windows object model

The Windows object model is important in both a conceptual and practical sense. Conceptually, it prescribes a way of thinking about and describing the fundamental objects that comprise any software application. In a practical sense, it defines the available objects and the relationships between objects. These relationships can be defined by users and developers programmatically in an object-influenced manner. Note my the use of the term "influenced" rather than "oriented", as the implementation of the object model at the user level is not defined as a strict object-oriented language.

Here are some definitions that will be useful throughout this discussion of the Windows object model:

OLE: The SAS System for Windows Enters the Bull Ring

- an object is defined as an item that can be programmed or controlled, such as a document, a text box, or pivot table.
- an object model is a representation—or conceptual map—of an application's functionality in terms of objects.

Each application included in Microsoft Office also has its own object model, which is defined similarly to the Windows object model. The Microsoft Excel object model is illustrated below in Figure 1. Each object model breaks the sets of objects and relationships into a hierarchy. The top level is the Application object, which has its own set of properties and methods. The Workbooks collection object, which consists of a series of individual workbooks in its collection, is the next level of the hierarchy, and so on.

This hierarchical object model design is ubiquitous throughout the Microsoft family of business products. As we will discover later in this presentation, the SAS System for Windows also has a object model.



OLE is a programming and inter-application communications framework that is integrated with the Windows object model. OLE is part of the operating system and is composed of two parts:

1. the **Component Object Model (COM)**, which is the underlying architecture for single workstations
2. a range of OLE services, including:
 - **application integration**: object linking and embedding, open editing, visual editing, and drag and drop
 - **custom solution development**: OLE automation and OLE controls
 - **enterprise integration**: OLE messaging, OLE DB, and Network OLE

The SAS System for Windows (Release 6.11, Wave 2) directly supports the application integration and custom solution development OLE services. We will visit both of these elements in this paper. The enterprise integration OLE services are still evolving.

Windows NT 4.0 will introduce a new version of COM, called Distributed COM (DCOM), which will connect objects across networks in a manner transparent to the user. In the same way that application partitioning splits the data and programming elements of an application across tiers of the computing infrastructure, DCOM promises to offer the same performance benefits for object-based applications. If DCOM delivers on these promises, it will represent a step forward for application development targeting the Internet and intranets.

Demonstration of application integration OLE services

This demonstration focuses on two of the earliest elements of OLE, including embedding and linking objects, as well as a more recent OLE development, called drag and drop. All three services are included in the OLE services for application integration.

Embedded and linked objects are examples of container services, a subset of application integration services, because they allow the user to place an object in a container, which itself is actually just another object. By placing objects inside other objects, we can create "compound documents," such as spreadsheet table objects inside word processing documents, or database query objects inside presentation slides. For example, the paper prepared for this presentation was composed in Microsoft Word, and the figures appearing throughout the paper were included in the paper via OLE container services.

The following is an excerpt from the excellent paper by Clegg and Rigsbee on OLE and the SAS System for Windows, presented at SUGI 21:

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. *[Author's Note: Holding the*

Ctrl key while dragging will not produce the desired result. In order to copy the object, you must only hold down the Ctrl key once it is "over" the container destination, just prior to releasing the mouse button to perform the drop.] 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 (Ø) sign.

One topic addressed in other OLE discussions is the application-specific support for drag and drop OLE services. Some applications, including Microsoft Word and the SAS System, do not support drag and drop linking. While the cursor may appear to perform a drag and drop link, you should check the object to verify that it has not been actually embedded, rather than linked.

OLE programming

As described above, objects in the Windows object model can be programmed or controlled. Direct control over object properties and methods is generally available by activating a dialog box or other type of menu directly accessible in the object.

Program control can take place at two levels. The first and more common level is the power user who connects pre-existing objects by way of a scripting or interpreted language. This level is much easier to learn, and the tool sets are available at generally lower cost. There are several languages available in Windows for accessing OLE services, including Visual Basic, Visual Basic for Applications¹, and the SCL language available in SAS/AF.

The more advanced level of OLE programming is at the level of the programmer who uses C++ and similar languages to build more complex software. Programming at this level has access to the OLE API. During a brief personal review of the OLE programming references (available from Microsoft Press and others), I rapidly categorized the people who can handle this level of programming as brave and brilliant!

Overview of the OLE/SAS Relationship

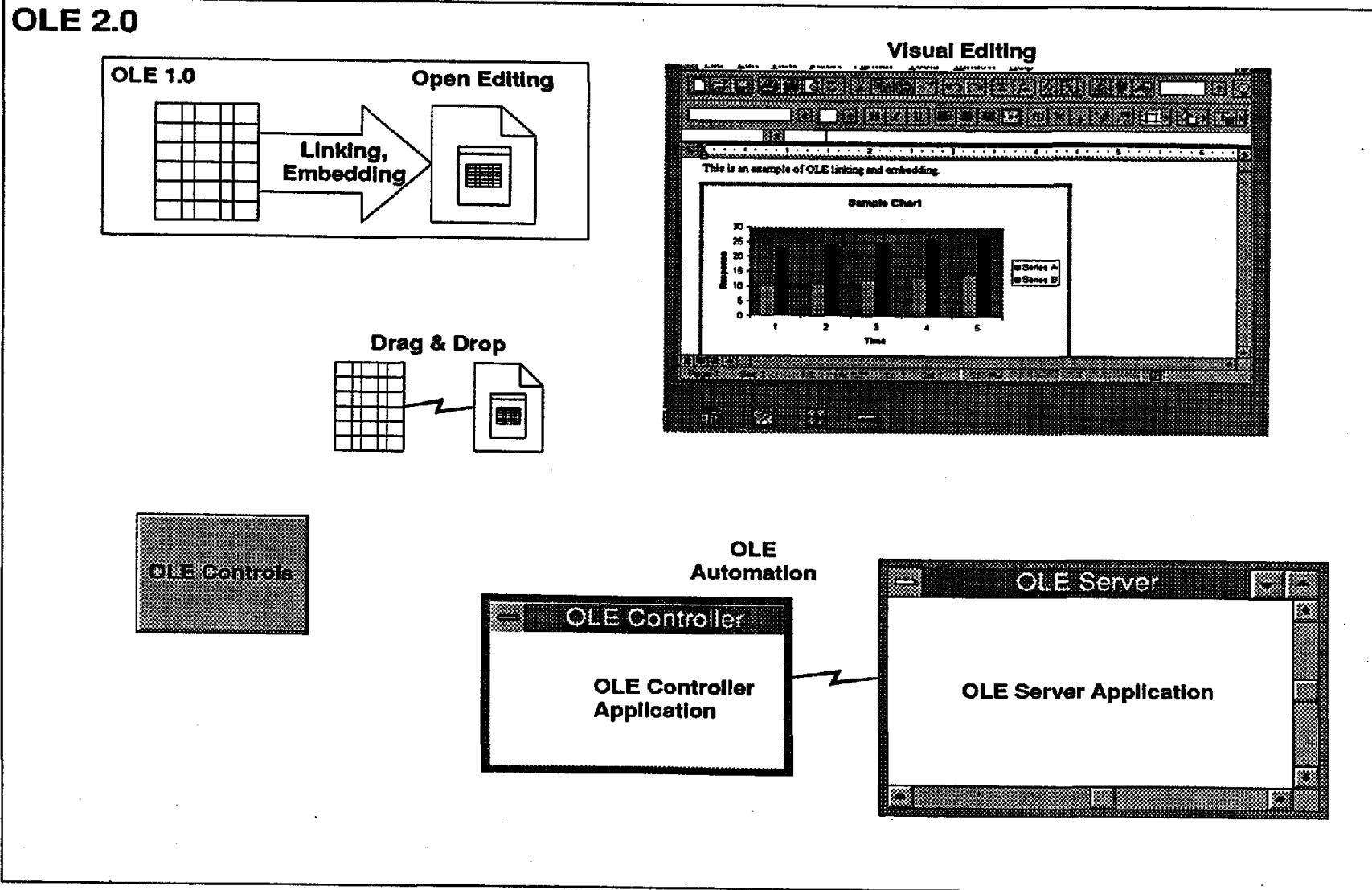
Previous releases of the SAS System for Windows (6.08 and 6.10) supported Microsoft's initial release of OLE 1.0. OLE 1.0 included support for object linking and embedding (from whence the acronym OLE was originally crafted). SAS Release 6.11 Wave 2 supports OLE 2.0, with support for OLE containers (embedded and linked objects, drag and drop, visual editing), OLE controls, and OLE automation.

OLE 2.0 maintains the OLE 1.0 functionality. Consequently, any SAS/AF applications from previous releases that took advantage of OLE 1.0 will continue to work in the SAS System for Windows Release 6.11. Figure 2 below illustrates the history of these OLE services.

¹ The VBA language is a superset of Visual Basic, and the object model for VBA is tuned to the Microsoft Office business suite, in which VBA is packaged. Microsoft recently licensed the VBA language to several major application developers for use in their products.

Figure 2

History of OLE Services Supported by the SAS System for Windows



The rest of this section will outline the relationship SAS has with these various OLE services. For greater detail on OLE services supported in SAS, please consult the references by Clegg and Rigsbee and by Corcoran included in the References section at the end of this paper.²

OLE Container Support

In addition to embedded and linked objects and drag and drop, OLE container support in Release 6.11 Wave 2 has been expanded to include visual editing. Visual editing services allow 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. (Clegg & Rigsbee, 1996).

An example of visual editing is included in Figure 2 above, where an Excel data range and chart have been embedded in a Word document, and being visually edited. Note that linked objects do not support visual editing; open editing (available in OLE 1.0) opens up a separate instance of the linked object's creator application, then returns control to the container application after the user completes the editing session.

Demonstration of Excel worksheet and chart objects embedded and linked in SAS/AF Frame entries

This demonstration focuses on linking and embedding objects inside SAS/AF Frame entries. Similar to the previous demonstration of embedding an Excel chart in a Word document, we can also embed or link objects in applications developed in SAS. Drag and drop is also supported, as is visual editing of embedded objects. Currently, there is no method for embedding SAS objects in other container applications, but this is changing. The final section of this paper discusses these changes briefly.

There are two general methods of embedding or linking objects in a Frame entry. The first is to use drag and drop, and the second is to use the pop-up menu and use one of the three OLE objects. When you save a Frame entry containing an OLE object, the OLE object will itself be saved with a catalog entry type of HSERVICE, separate from the entry in which it is contained.

The first pop-up menu, "OLE—Insert Object", is used to embed or link an object in a Frame entry, either one that exists in an external file, or a new object based on a list of applications on your desktop that support OLE 2.0. You can also create an OLE control; this is discussed in the next section of this paper.

The second pop-up menu, "OLE—Paste Special", is used to copy an object from the Windows Clipboard. This emulates the earliest (and possibly the simplest) means of embedding or linking objects, by copying and pasting. When using a Frame entry as a container, you cannot simply type Ctrl-V (i.e., to paste the object stored in the Clipboard) as you would in Excel or Word; instead, you use Paste Special to perform this task. If you use this option and do not subsequently rename the pasted OLE object by using the Object Attributes pop-up menu, it will be saved in the current SAS catalog with an entry name that follows the naming conventions for new objects in a Frame entry (e.g., OBJ1.HSERVICE, OBJ2.HSERVICE, etc.).

The third pop-up menu, "OLE—Read Object" is used to copy an OLE object from an existing SAS catalog.

I have personally found the pop-up menus to be nearly as fast as using the drag and drop method. Additionally, recall that the default action of drag and drop is to move the object from its server to the container. Drag and drop makes it quite easy to accidentally erase the object from its creator! Unfortunately, I didn't find any way to undo this behavior.

² The SAS/AF Software Frame Class Dictionary and the Changes and Enhancements to the SAS System for Microsoft Windows Environment, Release 6.11 also provide an excellent set of reference materials on this subject.

OLE Controls

OLE controls are very much like stand-alone OLE objects that act as embedded objects once included in an application. Formerly called OCXs (based on their default filename extension), OLE controls are an extension of objects designed by users and other third parties in Visual Basic (which in turn were called VBXs).³ OLE controls can be purchased from Microsoft and other vendors.

The SAS System for Windows Release 6.11 Wave 2 supports OLE controls, and includes two controls in the installed software (a combo list box object and a text entry box object). There are two constraints on the use of OLE controls in SAS Frame entries: they must be properly registered in the Windows Registry, and they must be 32-bit controls (not 16-bit).

Demonstration of OLE controls in a SAS/AF Frame entry

As mentioned above, OLE controls behave much like embedded OLE objects. This similarity persists in SAS. To add an OLE control to a Frame entry, use the AF Build facility's pop-up menu and select "OLE—Insert Object", and subsequently choose the radio box item labeled "Add Control." The dialog box will list all OLE controls that have been registered properly. Assuming SAS was installed properly, at least two OLE controls (the combo list box object and the text entry object) will appear. Clicking OK will place an instance of the selected OLE control in your Frame entry. You can also place OLE controls in your Frame entry by using the "OLE—Paste Special" option or by drag and drop.

Programming statements exist for every OLE control, and are accessible via SCL. Properties of each OLE control can also be set directly by clicking the right-hand mouse key on the OLE control. Since each OLE control has its own context-independent pop-up menu also, you may have some problems accessing the SAS Build facility pop-up menu. If this occurs, refer to SAS on-line help for the \$OLEUIDEAD option; this option disables the OLE control's user interface.

OLE controls also generate events that can be trapped in SCL code, similar to event trapping for native Frame entry objects. The Event Map dialog, available by invoking the Object Attributes dialog, allows you to set event handling code and the appropriate SCL source entry and label.

OLE Automation

OLE automation refers to the programmatic control of one Windows application's objects by another application, working via the OLE framework. The two sides to each OLE automation "conversation" are the automation controller, who initiates the conversation, and the automation server, who responds to the controller.

It is important to remember that it is the server's objects which are controlled by the automation controller. This was an initially confusing distinction to me, especially since the so-called "client-server revolution" seemed to make it clear that the server had all the power. But if you consider that the client makes requests of the server, which returns information to the client, then the confusion (at least partially) subsides.⁴

A more technically correct description of the automation server's role is that the server exposes OLE objects to automation. Each OLE object in turn exposes OLE verbs, which map to methods and properties of that OLE object. The automation controller's role is to seek out these OLE objects and their verbs, and to invoke methods and set properties of these OLE objects as it sees fit. This relationship is illustrated in Figure 3 below.

³ OLE controls improve on VBXs by adding ambient properties (allowing communication between the container and the OLE control) and events (actions generated by the OCX to the container, such as a mouse click or highlighting).

⁴ Or much simpler: If all you remember is that the controller is the one that does the controlling, then the other half becomes germane!

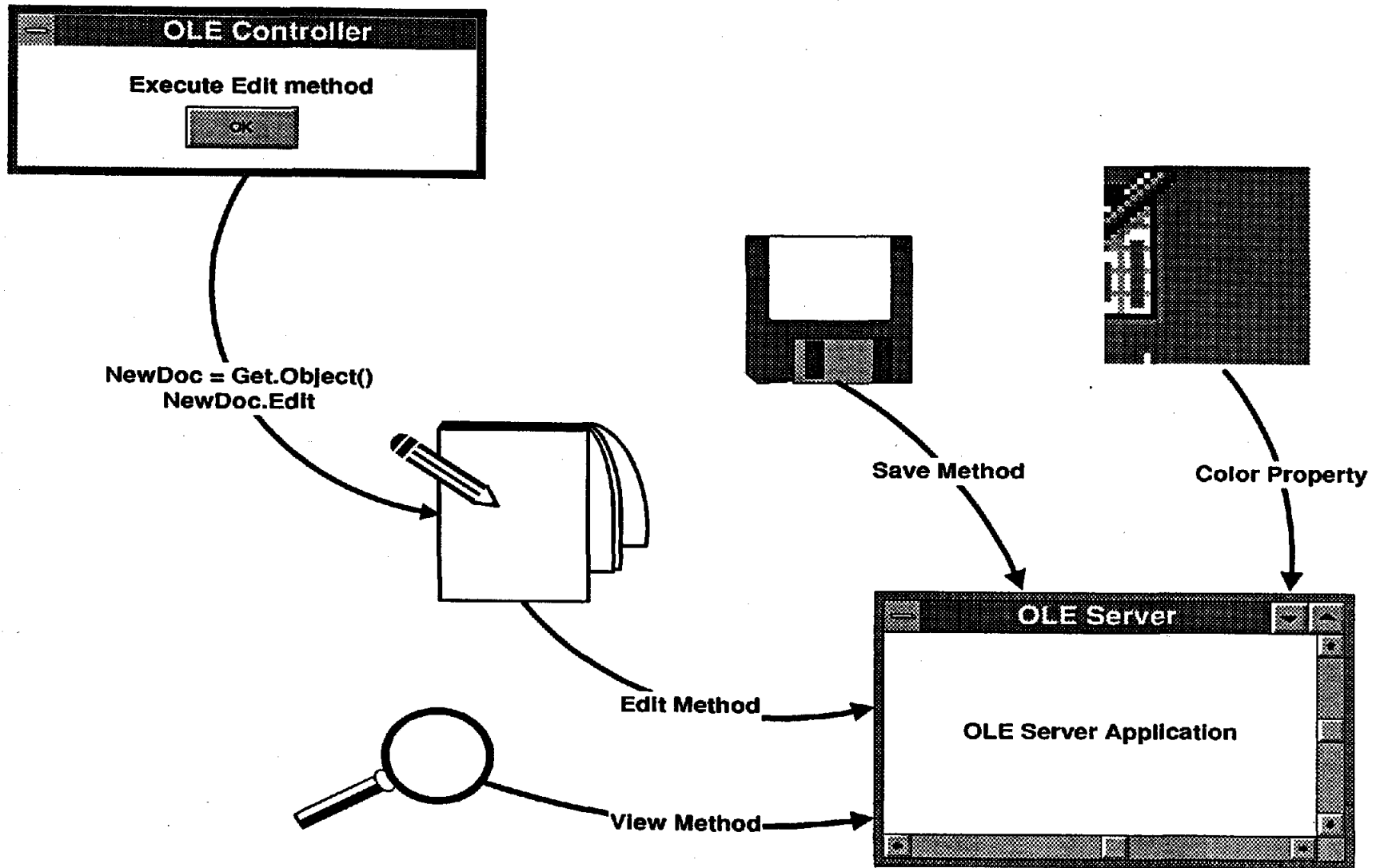
OLE: The SAS System for Windows Enters the Bull Ring

The bottom line to me is that OLE automation is an incredibly exciting technology that promises to blur the lines between application environments contributing to development of custom applications. It means that SAS/AF (and its family of objects), Visual Basic, third party OLE objects, and a host of other application development environments that operate under OLE 2.0 standards can be used to combine selected objects and languages of the developer's choice. It doesn't mean that developers won't have to struggle with a new breed of problems related to this sort of "mix-and-match" or "best-of-breed" strategy. But OLE automation does open doors, particularly to SAS/AF developers.

SAS operates both as an automation controller and as an automation server. As a controller, SAS uses SCL code to invoke OLE verbs of exposed OLE objects, such as business applications from Microsoft Office, or PowerBuilder, or Oracle Power Objects, etc. An example of SAS controlling Microsoft Excel appears later in this paper.

Figure 3

Comparison of Automation Controllers and Automation Servers



The SAS automation server

As an automation server, SAS exposes one OLE object, which is essentially the SAS System itself.⁵ However, at SUGI Carol Rigsbee assured us that SAS Institute is planning to develop greater functionality in this area, and that SAS Institute is open to our feedback about what kinds of OLE automation objects should be developed.

The SAS System automation object currently has five methods and eight properties, which appear in Tables 1 and 2, respectively. These methods and properties allow the developer to programmatically obtain access to the SAS System command line and procedure submittal mechanisms (i.e., to be able to submit SAS jobs). These OLE verbs can be invoked from any OLE 2.0-compliant application environment; Visual Basic is one of the most popular. Typical OLE automation commands specific to the SAS automation server written in Visual Basic code appear in both Tables 1 and 2. A more complete description of these methods and properties appears in the excellent paper "The SAS System OLE Automation Server," by Brian Corcoran, which is available on request from SAS Institute.

Table 1
SAS Automation Server Methods

Method	Parameters	Returns	Visual Basic example
Command	String	nothing	SAS.Command("build")
QueryWindow	String	True/False	Exists=SAS.QueryWindow("Log")
Quit	none	nothing	SAS.Quit
Submit	String	nothing	SAS.Submit("data; x=1; run;")
Top	none	nothing	SAS.Top

Table 2
SAS Automation Server Properties

Property	Default Value	Read/Write	Visual Basic example
Busy	None	Read	if (SAS.Busy) then...
CommandWindow	"" (empty string)	Read/Write	SAS.CommandWindow = "Program Editor"
CommandWindowVisible	True	Read/Write	SAS.CommandWindowVisible = True
RC	0.0	Read	if (SAS.RC = 0) then...
ResultString	""	Read	message = SAS.ResultString
Title	"SAS"	Read/Write	SAS.Title = "My App"
Visible	False	Read/Write	SAS.Visible = True
Wait	False	Read/Write	SAS.Wait = True

⁵ At the time of the current conference, many sites are running Wave 1 of the SAS System for Windows Release 6.11. The SAS OLE automation server was not supported until Wave 2, which was shipping at the start of June 1996. Fortunately I received our Wave 2 CD-ROM just in time to prepare the demonstrations of this feature for the conference!

To create an instance of an OLE automation server from an OLE controller, you need to refer to the server application by a very specific name, which is registered in the Windows Registry. The name for the SAS System for Windows Release 6.11 Wave 2 application is "SAS.Application.61102." In Visual Basic, you use the `CreateObject()` and `GetObject()` functions to create and refer to OLE automation objects, respectively. The function creates a nickname, much like a SAS filename, to represent this OLE automation object. For example, the Visual Basic code examples in Tables 1 and 2 above refer to the SAS OLE automation server as an object simply called "SAS." The VB code necessary to create this reference would be the following:

```
Set SAS = CreateObject("SAS.Application.61102")
```

An important topic to bear in mind when developing your first SAS OLE automation server application is termination of the OLE automation session. Unless you pass the proper commands to the automation server (SAS in this case), it will stay active until the system is shut down.⁶

Demonstration of Excel and VBA controlling a SAS automation session

I adapted a demonstration of Visual Basic acting as an OLE automation controller (Corcoran, 1996) to Visual Basic for Applications. While the OLE automation logic is largely the same, I have dropped the code related to controlling native Visual Basic objects, in order to simplify the demonstration. The VBA code for this demonstration appears at the end of this paper.

Demonstration of SAS SCL controlling an Excel automation session

I adapted a demonstration of SAS acting as an OLE automation controller (Clegg & Rigsbee, 1996) to Microsoft Excel. I modified the `ChartWizard` method arguments to create a 3-d column chart with more specific axis labels and no legend, but the rest of the logic is equivalent.

ActiveX and the Future of OLE

Re-branding of the OLE product line

Microsoft chose recently to re-christen the OLE framework with a new name: "ActiveX."⁷ Computing media pundits believe that Microsoft may create confusion in the market about the nature of ActiveX and how it is or is not similar to OLE (when in fact they are identical). Table 3 below identifies the past, present, and future of the OLE/ActiveX naming conventions. (Leach, 1996)

⁶ Note that in Visual Basic, the OLE verb "Quit" invoked as the SAS OLE method `SAS.Quit` will work fine. However, `Quit` is a reserved keyword in Visual Basic for Applications (VBA). Two acceptable alternatives to the `Quit` verb are available. You can use either of the following approaches:

```
SASobj.Command ("endsas")
```

or

```
SASobj.Submit ("endsas;")
```

Please refer to the code samples at the end of this paper for a complete example..

⁷ I'm sure they had their reasons well steeped in market research; one colleague tells me that the use of "X" in the name "ActiveX" is supposed to make us think that OLE and Windows is inherently open, by establishing a connection (however tenuous) with the brand name for the flavor of Unix called "XWindows." My personal guess is that "Active" sounds more humanized and "X" sounds more mysterious than "OLE"—who wants a bunch of mad, snorting bulls leaping out of their spreadsheet, anyway?

Table 3

Development of the OLE/ActiveX Name Brand

Past	Present	Future
Object Linking and Embedding	OLE Interfaces	ActiveX
OCX	OLE Controls	ActiveX Controls
OLE Automation	OLE Automation	ActiveX Automation
Network OLE	DCOM	Undetermined

Source: Leach, 1996

The history of OLE suggests to me that such a name change may not be such a bad idea anyway. Many people may be inclined to think of OLE as merely "object linking and embedding," when really the technology has surpassed these boundaries. In addition to OLE controls and OLE automation, lesser known developments have also donned the OLE mantle, including OLE DB, an upper layer to data translation through ODBC, and OLE Transactions, a transaction-management technology.

Microsoft's Nashville release

At the recent Internet developers conference in San Francisco, Microsoft unveiled "Nashville," which promises to bring Internet and intranet features closer to the core of the Windows operating system. Nashville is a Windows shell that integrates a Web browser, a disk browser, and business software such as Word and Excel. This shell should be available in the 3rd Quarter of 1996 as an upgrade to Windows 95 and Windows NT 4.0. (Honeyball, 1996)

ActiveX (OLE) controls will be major components of the Nashville interface (as well as Java compatibility). The key improvement that Nashville offers is that it joins the HTML system (for Web page construction) with the native file formats used in other applications. This means that moving around in Windows Explorer from a Web page to an Excel spreadsheet to a Word document will only involve a matter of pointing and clicking. The editor will handle all of the inter-application translation. This will most likely replace the open and visual editing services offered in the current version of OLE. This also helps us do away with having separate browsers and editors for text files, word processing, spreadsheets, presentations, databases, and so on.

SAS Viewer

The browsing and editing improvements offered in the Nashville release of Windows (see above) are related to a recent development available in the SAS System for Windows 95, called the SAS Viewer. This technology allows any user of Windows 95 to view the contents of SAS datasets and other native SAS objects without having the SAS System installed on their desktop. This technology is available in Wave 2 of Release 6.11 for Windows, and is integrated with the Windows 95 Explorer.

SAS OLE objects

I mentioned above that the only OLE object available in the SAS System for Windows was the system itself. However, SAS Institute is in the process of developing prototype OLE objects based on native components. At SUGI 21, I saw a prototypical OLE object based on a SAS/Insight rotating 3-D scatter plot embedded in Word. The prototype is still too preliminary to release to the public (I asked). And you can't simply do it yourself by copying an instance of a rotating scatter plot to the Clipboard and then pasting into Microsoft Word (I tried—all you get is a static picture—no rotation, no nothing).

Soon, however, OLE objects based on native SAS components will be introduced, perhaps in Release 7. This will offer SAS application developers yet greater flexibility and power in composing custom applications across a variety of development environments.

Conclusion

OLE is a fundamentally important construct underlying the most popular desktop operating system today. The future relationship of the SAS System for Windows to OLE is an important one for users and developers alike to understand. As we move from visual editing and drag and drop, to OLE automation to ActiveX controls, to the Nashville shell and SAS OLE objects, our flexibility and power as users and developers will increase dramatically.

References

- Clegg, Jennifer, and Rigsbee, Carol. "OLE and the SAS System for Windows Release 6.11", in Proceedings of the Twenty-First Annual SAS Users Group International Conference. Cary, NC: SAS Institute. 1996.
- Corcoran, Brian. "The SAS System OLE Automation Server", in Proceedings of the Twenty-First Annual SAS Users Group International Conference. Cary, NC: SAS Institute. 1996.
- Febish, George J., and Sarna, David E.Y. "OLE and Development Tools", in Microsoft Solutions Developer's Kit 2.0. Redmond, WA: Microsoft Corporation, 1996
- Honeyball, Jon. "The Road to Cairo Goes Through Nashville." Windows NT Magazine, June 1996, pp. 37-40.
- Leach, Norvin. "ActiveX branding stirs confusion." PC Week, 13 (21), May 27, 1996, p.15.
- Rigsbee, Carol. "Introducing Release 6.11 of the SAS System for Personal Computers", in Proceedings of the Twenty-First Annual SAS Users Group International Conference. Cary, NC: SAS Institute. 1996.
- SAS Institute. "SAS Institute Annual Report 1995." Cary, NC: SAS Institute, 1996.
- SAS Institute. Changes and Enhancements to the SAS System for Microsoft Windows Environment, Release 6.11 Cary, NC: SAS Institute, 1995.
- SAS Institute. SAS/AF Software Frame Class Dictionary Cary, NC: SAS Institute, 1995.

Code Examples

SAS OLE automation server controlled by Excel and Visual Basic for Applications

Visual Basic for Applications (VBA) is a specialized superset of Visual Basic, tuned to the Microsoft Office object model. In this demonstration, Excel is used as the OLE controller, and VBA code is the script that programmatically invokes OLE verbs exposed by the SAS automation server.

All three of the following subroutines are connected to simple push buttons dropped onto an Excel worksheet. It is critical to use the Public variable declaration statement at the top of the VBA code so that all subroutines will recognize the existence of the SAS OLE automation server object.

The first subroutine (StartSAS) starts the SAS OLE automation server, makes the session visible, sets the title of the session, runs a data step and Proc G3D, and copies the contents of the default window (the Graphics window, in this case) to the Windows Clipboard.

The second subroutine (PasteChart) pastes the contents of the clipboard to the same worksheet, which resolves as an Excel Picture object displaying the familiar Cowboy Hat graphic.

The third subroutine (quitSAS) closes the SAS OLE automation session. Please note that you cannot use the OLE verb "Quit" if you are using VBA as your automation controller. "Quit" is a reserved keyword in VBA, but not in VB. Since VBA native keywords apparently take precedence over OLE verbs, this statement does not execute properly. The proper alternatives are either to use the Command verb to send the "endsas" command to the SAS command line, or to use the Submit verb to send the "endsas;" statement to the Program Editor.⁸ It is also important to use "Set SASobj = Nothing" in order to let the SAS OLE automation session know that it is okay to close.

```
Public sasobj As Object

Sub StartSAS()
    Set sasobj = CreateObject("SAS.Application.61102")
    sasobj.Visible = True
    sasobj.Title = "Hat Graph"
    sasobj.submit ("data hat; do x=-5 to 5 by .25;")
    sasobj.submit ("do y = -5 to 5 by .25; z=sin(sqrt(x*x+y*y));")
    sasobj.submit ("output; end; end;")
    sasobj.submit ("proc g3d data=work.hat; plot y*x=z/ctop=red;")
    sasobj.submit ("title 'Cowboy Hat'; run; quit;")
    sasobj.command ("gstore")
End Sub

Sub PasteChart()
    Range("A10").Select
    ActiveSheet.Pictures.Paste
End Sub

Sub quitSAS()
    sasobj.command ("endsas")
    Set sasobj = Nothing
End Sub
```

⁸ Many kudos to the superb Yvonne Selby in SAS Institute's Technical Support Division for finding these solutions!

Excel OLE automation session controlled by SAS

In this demonstration, SAS is used as the OLE controller, and SCL code is the script that programmatically invokes OLE verbs exposed by the Excel automation server. An instance of the SAS/AF Push Button class runs the SCL code in the section labeled "BUTTON."

I modified the ChartWizard method arguments to create a 3-d column chart with more specific axis labels and no legend, but the rest of the logic is equivalent. Note that the reference to the constant -4100 is the Excel constant "xl3DColumns."

```
/** c:\p\pnwsug\conf96\oleauto.scl ***/

length rangeid $ 80;

BUTTON:

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

/* open worksheet */
call send (excelobj, '_GET_PROPERTY_', 'Workbooks', wbsobj);
call send (wbsobj, '_DO_', 'Open', 'c:\p\pnwsug\conf96\sales.xls');
call send (excelobj, '_GET_PROPERTY_', 'ActiveSheet', wsobj);

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

/* get the range of cells to chart */
call send (wsobj, '_COMPUTE_', 'Range', 'A2', 'B5', rangeobj);
call send (rangeobj, '_GET_REFERENCE_ID_', rangeid);

/* chart the cells */
call send (chartobj, '_DO_', 'ChartWizard', rangeid, -4100, 1,
2, 1, 1, 2, 'Automation Demo!', 'Year', 'Sales', '');
return;

QUITXL:
/* close the Excel OLE automation session */
call send (excelobj, '_GET_PROPERTY_', 'ActiveWorkbook', awbobj);
call send (awbobj, '_DO_', 'Close', 'False');
call send (excelobj, '_DO_', 'Quit');
call send (excelobj, '_TERM_');
return;
```