

**Prepared By: George J. Febish of ObjectSoft Corporation**  
**Last Update: June 30, 1993**

**OLE 2.0 incorporates both Object Linking and Embedding from which it got its name and OLE Automation**

### **Object Linking and Embedding**

**Object Linking and Embedding** is the most misunderstood part of Windows 3.1. Vendors are spending a lot of time and money implementing the OLE strategy and still have drastically different looks and feels. Microsoft has published a design guide, entitled, with the Windows 3.1 SDK that discusses the standardization of OLE implementations. Unfortunately even Microsoft has a problem keeping to the standards as can be seen in Word for Windows 2.0's implementation.

OLE can be thought of as a complex Windows API call involving many actual calls with several complicated data structures. DDE was also complex, although nowhere near as complex as OLE, and it was implemented in most VDEs as 1 or 2 lines of code. For the VDE programmer, OLE needs to be the same.

More confusion exists between the role of DDE and OLE. OLE is built on top of DDE. It uses the DDEML libraries to implement its communications within Windows. Possibly the two will merge in the future or OLE will take on more of the DDE capabilities but as they stand today we need both. The Visual Basic R 2.0 Programmer's Guide described the differences best:

*"The advantage of using OLE in this type of application is that you don't have to do any work to display the linked data's image. The image appears exactly the way it is in the original spreadsheet."*

*"The major disadvantage to linking with OLE is that it can be difficult to access the linked data programmatically. If you need programmatic access to linked data, DDE is a better solution. If you want an easy way to display linked data, or want to allow the user to edit that data in its native environment, then use OLE."*

OLE 2.0 Automation changed this since the underlying data is easily made available (see OLE Automation section below).

OLE produces compound documents which consist of parts built by separate Windows applications. Both Linking and Embedding appear similar to the user, with linking preserving only information on the link and the actual data is placed in a file. Embedding preserves the actual data in the compound document.

When a user requests an edit of a linked document, the link information is sent to the OLE Application to load the correct file for editing.

When the user requests an edit of an embedded document, the actual data is passed to the

Corporation

OLE Application for editing (no file is involved).

OLE as a Container Application is some what easier a task to be provided by the VDE programmer. The OLE Application is still very complex and costly to provide in this fashion. Visual Basic and PowerBuilder both implement an OLE Container implementation as an object with properties, Events, and methods. PowerBuilder only supports OLE 1.0 Applications while Visual basic supports OLE 1.0, OLE 2.0, and OLE 2.0 Automation Applications.

Perhaps OLE can be best understood to us mere mortals by reviewing the Visual Basic OLE 2.0 control included in the new Visual Basic R 3.0.

Visual Basic OLE 2.0 Control

The OLE control lets you display data from another Windows application (OLE Application) in your Visual Basic application (OLE Container Application), and allows the user to edit that data from within the OLE application that it was created. OLE also allows the sending of verbs (commands) to the OLE Application for execution of services.

The OLE 2.0 concepts:

**OLE Objects** - Data from a OLE application that can be displayed and manipulated through the OLE Container application (OLE Application). This object contains the name of the OLE application, its data or a linked reference to the data, and an image of the data.

**Class** - identifies the OLE application that provides the data. and the type of data the object contains.

**Container Application** - An application that receives and displays the object's data.

**In-Place-Editing** - OLE 2.0 supports a concept of in-place-editing if the OLE Application supports this new protocol. Normally OLE initiates a new instance of the OLE Application, in it's own Window, to manipulate or create the OLE data for the OLE Container Application. OLE 2.0 can still do this but can also allow the OLE Application to take over control of the OLE Container application's Window and menu bar to provide an in-place-editing facility.

**Linked and Embedded Objects**

An object is either linked or Embedded but not both at the same time in the same OLE Container Application's Object. A linked object contains a place holder that is a reference to the data in a file readable by the OLE application. Embedded data is actually contained within the OLE Container Application's object and not in a file. OLE Container Applications can write this data to a DOS file or DBMS for safe keeping accross instances of the application. This DOS file is known only to the OLE Container

## Corporation

Application and should not be confused with the linked file that is readable by the OLE application.

Linked objects allow access to the underlining data through the OLE Application by many different people. When the OLE Container application starts it gets the latest copy of the data from a OLE Application and displays it in the OLE Container Application's control. This is usually used when the data in the link is used outside the context of the OLE Container Application's compound document. i.e. A sales department maintains a sales forecast, actual sales, and deviations in an Excel spreadsheet. An executive prepares a slide presentation for the Board of Directors and OLE links this data into one of his/her slides. When a user edits a linked data, the OLE Container Application sends the name of the linked file to the OLE Application for loading and editing. After editing the file is saved for safe keeping and a new image of the modified data is returned to the OLE Container Application for display.

Embedded links are used when the data is never used outside the compound document. The OLE Embedded data makes the OLE Container Application's document a complete application including its own data as well as the data needed to rebuild the OLE objects. When the user edits this type of object the OLE Container Application sends the only copy of the data that exists to the OLE Application for editing. The OLE application edits the data but does not store it in a disk file, it is maintained in memory. After editing is complete the modified data and its image is returned to the OLE Container Application for displaying and safe keeping.

### **OLE Properties**

An OLE Container has several properties that effect its operation:

**Action** - Determines an OLE operation to be performed on the OLE object by the OLE Container Application i.e. saving it's data to a file (see next section). Do not confuse this property (Method) with the OLE Verbs which are actions to be performed by the OLE Application on the object like Play, Edit, delete.

**AppIsRunning** - Idicates if the OLE Application is already running.

**AutoActivate** - Activates the OLE Object when user Dbl-Clicks on OLE Object without any VB code.

**AutoVerbMenu** - Activates the OLE Object verb pop-up menu when the user clicks the right mouse button over the OLE Object.

**Class** - Determines the type of data the object contains and the name of the OLE application associated with the data. This information is kept in an OLE registration database on each system. OLE Container Application products that support OLE 2.0 must register themselves with Windows. Registration includes the Class name, OLE

Corporation

application name, verbs supported, and default verb executed when the user double-clicks on the OLE object.

**DisplayType** - Determines if the OLE object displays an icon or the image of the data provided by the OLE Application.

**HostName** - Sets the OLE displayed name of your application while in the OLE Application.

**ObjectVerbFlags** and **ObjectVerbs** - Contains Flags and verbs supported by the OLE Application for the Object loaded. Flags determines for each verb its state (Greyed, checked, etc.) The verbs can be loaded onto a menu for selection by the user. To update this list issue Action=17 (fetch Verbs). The VB OLE 2.0 control automatically builds a pop-up menu over the OLE Object with each verb listed. Press the right mouse button over the OLE Object to get this menu.

**OLEType** - Type of OLE Link in OLE Object:

- 0      Linked
- 1      Embedded
- 3      None

**PasteOK** - Indicates that the clipboard contains an OLE object.

**SizeMode** - Determines how the OLE control is sized:

- 0      Actual size but is clipped if                      larger than object size
- 1      Stretches to fill control
- 2      Autosizes the OLE control

**SourceDoc** - Used to identify the file used to create the OLE object.

**SourceItem** - Used to identify the data (container) within the SourceDoc to be linked. i.e. a range of cells in a spreadsheet.

**UpdateOptions** - Determines how the OLE Object will be updated when linked data is changed.

- 0      Automatically updated
- 1      Frozen until OLE Application does a save
- 2      Manually updated by Action=6 (Update)

**Verb** - Determines action to be taken when OLE Object is activated (Action=7).

- 0      Default Action
- 1     Edits Object (supports in-place-editing)
- 2     Edits in its own Window
- 3     Embedded Objects - hides OLE Application from user

Remember if you Embed an object you are responsible for saving and loading data into

Corporation

the OLE Object at OLE Container application's termination/load time. A set of I/O functions is provided by the OLE Object for this purpose. The OLE Object's Action Property performs these functions as well as other operations on the object:

**CreateEmbed(0)** - call the OLE Application to create a new embedded object.

**CreateLink(1)** - **Creates an OLE Linked object from the OLE Application's file.**  
2 and 3 reserved for future use.

**Copy(4)** - Copies the linked or embedded object to the Windows clipboard.

**Paste(5)** - Pastes a linked or embedded object from the Windows clipboard. Check the PASTEOK property for TRUE first. Used to implement Edit-Paste menu command. if the paste was successful the OLEType property will be set to 0 (Linked) or 1 (Embedded).

**Update(6)** - Retrieves current data from OLE Application

**Activate(7)** - Activates the OLE Application with the verb in the Verb property.

*8 is reserved for future use*

**Close(9)** - Closes the OLE Object's link to the OLE Application and terminates the OLE Application. Equivalent to choosing Close from the Objects Control Menu.

**Delete(10)** - Deletes an OLE Object

**SaveToFile(11)** - Saves the OLE Object's data to a file.

**ReadFromFile(12)** - Reloads the OLE Object with the previously saved OLE data (using the OLE\_SAVE\_TO\_FILE)

*13 is reserved for future use*

**InsertObjDialog(14)** - Displays the Insert Object Dialog Box. Allows the user to create an OLE object and type. Use the OLETypeAllowed property to limited types.

**PasteSpecialDlg(15)** - Displays the Paste Special Dialog Box. Allows the User to paste a clipboard object and choose the Type of OLE Object to create. Use the OLETypeAllowed property to limited types.

*16 is reserved for future use*

**FetchVerbs(17)** - Updates the objects set of verbs supported by the OLE Application.

**Convert(18)** - Saves an OLE 2.0 object in an OLE 1.0 format.

Corporation

## OLE GUI Standards

The Edit Menu is used to interface the OLE Container Application and OLE Application together. The commands typically used are:

- Copy** - Used to copy an OLE Object from an OLE Application to the Clipboard
- Paste** - Used to paste an OLE object on the clipboard into your OLE Container application as an embedded object.
- Paste Link** - Same as Paste except the OLE Object is Linked.
- Paste Special** - Opens the Paste Special Dialog Box and allows user to choose Paste (Embed) or Paste Link (Link) type of OLE Object.
- Insert Object** - Creates a new embedded object in the OLE Container Application by allowing the user to select the OLE Class from a list of classes in the OLE registration database. After selection the OLE Application is called to create the new object.

## OLE 2.0 Menu commands

You should support an Edit menu similar to the one below

**Object** - Names the OLE object in the OLE control and provides all verbs on a cascading menu

**Cut** - Copies OLE object to clipboard and deletes it from OLE Object

**Copy** - Copies OLE object to clipboard

**Paste** - Pastes clipboard OLE object as an embedded object type.

**Paste Special ...** - Displays Paste Special Dialog Box for user to choose type of OLE object. It also informs user of object on clipboard.

**Delete** - Deletes the OLE object.

**Update** - Calls the OLE Application to update the data in the OLE Object.

## OLE 2.0 Automation

A standard facility for an OLE Application to expose its object's properties and methods for use in a remote OLE Container Application. Exposed OLE Automation Objects react as if they were local objects in a VDE. You typically read and set properties and cause methods to be invoked. The following is an example in Visual Basic 3.0 using a

Corporation

hypathetical sp[readsheet OLE Application:

```
Dim spdS as Object  
Set spdS = CreateObject("spdSheet.Worksheet")  
spdS.Row = 16  
spdS.Column = 1  
spdS.Insert = "6700"  
spdS.SaveAs App.Path + "\MySheet.SPD"
```

Expect to see OLE 2.0 Automation in every Microsoft product. The "Glue" that holds OLE 2.0 together (Common Macro Language) is Visual Basic in a new Edition called the Application Edition (VBA). Expect to see it first in Excel and Powerpoint 3-4th quarter and Word first Quarter 1994.

ObjectSoft will release a book on this exciting topic from Bantam in first Quarter 1994.

Corporation