Corporate Backgrounder

Object Linking and Embedding (OLE) in Microsoft® Windows™ Operating System Version 3.1

An Overview of Data Sharing Technology in Windows 3.1

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Executive Summary

Object linking and embedding (OLE) is a major feature of Microsoft® Windows™ version 3.1 that lets you achieve real application integration. With OLE, Windows 3.1 can enhance the way you work with your computer.

What is OLE?

OLE lets you combine information from a number of different applications into a single document. OLE is:

- **Powerful.** It lets you easily create documents that incorporate information from a variety of sources. OLE also gives you an easy way to link data that resides in more than one application. Information that is made current in one application will be automatically updated in all other linked applications.
- **Easy to use.** You just choose a simple menu command to integrate data from one application into another application, and double-click to edit that information.
- **Fully integrated.** OLE functionality is built into the Windows operating system so there's nothing extra to obtain or install. OLE is an open standard, which means anyone developing applications for Windows can include OLE capabilities. The standard includes guidelines for how the OLE user interface should work, so procedures for using OLE will be similar from one application to the next.

Benefits of OLE

OLE offers a number of key benefits to people who work with compound documents. You can:

- **Organize your work.** When you embed objects in a document, they're all stored in the same file as the document itself, so they're easy to keep track of. That's especially useful when you want to send someone a document that consists of many independent pieces. You're also freed from the tedious jumping back and forth between applications as you try to put your documents together.
- **Use the best applications.** You can choose the best application for creating each part of your document—now, and in the future. OLE client applications can store any data format—even ones that haven't been invented yet.

- **Stay current.** Linking lets you include data from other applications in your document without greatly increasing its size. Several documents can be linked to a single object, so changes in the object appear in all of the documents. That’s a good way to make sure different users stay in sync with the latest data.

Table of Contents

µ

The Evolution of Shared Information

Before we go into a more detailed look at OLE, it might be helpful to understand the evolution of moving and sharing information among applications on PCs that eventually led to the OLE concept. Soon after people discovered the benefits of personal computers, they also discovered one of the limitations: If you wanted to bring together different kinds of data such as text and graphics, the only way to do it was to print all of the files individually and then produce a compound document with scissors, tape, and a photocopier. Then things started to improve.

First, there was electronic cut and paste

The first steps beyond scissors and tape were the electronic cut-and-paste capabilities found in the Windows environment. The typical procedure was to create text, a drawing, a spreadsheet, or other piece of information in one application, then electronically cut or copy the text, drawing, or spreadsheet you wanted to the Windows Clipboard, which provided the basic means for moving data between applications. Then you would open your target document in a second application and paste the text, spreadsheet, or drawing where you wanted it. Your final document then contained a static bitmap or metafile image of the information created in the other application.

The disadvantage of the electronic cut-and-paste process became clear when you needed to revise the information you pasted into your document. You had to find the right file (if you had the foresight to save the original version of what you pasted into your document), load it into the application that created it, make the changes, and recopy the revised version into your final document. If you didn't save the original, you started over from scratch.

Then, there was dynamic data exchange

Dynamic data exchange (DDE) was the next step in the evolution of moving data from one application to another. With DDE, you can create an electronic link between two files (for example, you can link a Microsoft Excel chart to a Microsoft Word for Windows document). When the chart in Microsoft Excel is changed, that change also appears in the Word report where the chart has been pasted. This frees you from having to remember changes in one document that affected information in others.

DDE has some limitations, however. With DDE, you get the data from the Microsoft Excel spreadsheet chart in your Word document, but not necessarily the formatting from Microsoft Excel. It's up to the receiving, or client, application (Word) whether it will be displayed correctly. For that reason, the client application must be able to understand the data it gets via DDE from the other application.

Another limitation of DDE is that it's only good for bringing data into a document. It doesn't give you an easy way to get at the application that created the imported data or the original source file so you can quickly make changes. You still need to open the second application and find the source file to make the adjustments you want. (For a more detailed look at the features in OLE that DDE did not support, see Appendix B.)

Now, there's object linking and embedding

OLE is the latest step in the evolution of application integration. By using OLE to embed information, you can combine text, charts, spreadsheet sections, even voice instructions (if you have the right hardware) into a single document, regardless of the data format. OLE lets you instantly call up the application that created the information from within your new document so you can change embedded information "on the fly." With OLE, the data you are combining can actually be stored in the file, so you don't have to find the original source file before making changes.

µ §

Some OLE Fundamentals

There are a few fundamental terms and concepts that will help you better understand OLE technology. The following definitions will be used in this document.

Server

An application that creates information to be linked or embedded into another application. Server (or source) applications also perform operations (such as editing or playing) for some types of OLE objects such as spreadsheet charts. Microsoft Excel would be the server application for a spreadsheet chart that you wanted to embed in a Word document.

Client

An application or document that receives, stores, and displays objects created by server applications. If you use Word to create a report and then import your spreadsheet chart from Microsoft Excel, Word would be the client (or container) application in the exchange.

Object

Anything that can be linked or embedded in a client document, such as a drawing from Microsoft Paintbrush™, a spreadsheet from Microsoft Excel, or a piece of digitized sound. Even an MS-DOS® operating system–based command line can be an object.

Compound documents

These are documents that contain data (in different formats) from more than one application. A compound document is put together in an OLE *client* application, such as Word for Windows. That document can include linked or embedded *objects*—a spreadsheet, a drawing, a voice recording, or the output of any other Windows-based application.

Embedding

You can think of embedding as a more powerful version of electronic cutting (or copying) and pasting between applications. In both cases, the data that makes up the object is moved from the server application and stored in the client document. But there are *two major differences* between embedding and electronic cutting and pasting:

1. The client application doesn't have to understand the data in the embedded OLE object; in fact, it doesn't even have to access it. The object may contain an image of the data that the client application can display, or (for sound recordings, for example) it may contain an icon that marks the object's place in the document.

2. When you double-click the image or icon, the Windows operating system starts the *server* application. You can use the server application to edit the object. When you quit the server application, you return automatically to the compound document. The image of the object in the compound document is updated to reflect the changes you just made.

Embedding means new possibilities for data exchange. For example:

- When you send the compound document electronically to someone who has a multimedia-capable machine, they can double-click an icon to hear a sound recording or see a video sequence. The server application starts, plays the sound or video, and then quits and returns them to the compound document.

- You can electronically mail someone a report and if the person receiving the report has the same applications you do, they can view and change information you have embedded in the report, then electronically mail it back to you.

- OLE makes information highly portable. If you aren't on an electronic mail system, you can give someone a report on a floppy disk. They can access information (a spreadsheet, for example) that you have embedded in the document, make changes, and give it back to you. There are no links to break, and they don't need to gain access to a source file on your machine in order to make edits.

Advantages of embedding include the fact that you don't need separate source files in order to pass information among documents or among

members of your workgroup. Documents become totally self sufficient.

Some disadvantages of embedding include the fact that file sizes get larger when you embed information in them. In addition, information must be manually updated, since the information is embedded in the document and not linked to a source file.

Linking

Linking in OLE is similar to DDE. Unlike embedding, with linking you store a separate source file for the data you want to share with other applications, and a pointer (path) to that source file goes into your document. Documents are linked so that when the source file changes, all the documents linked to that source file change as well.

These link updates can be either automatic or manual. If your client document (your Word document, for example) is open, automatic links are updated when you make changes to the Microsoft Excel chart that is linked to the Word document. If the client document is closed when the changes are made, you will be asked when you open the document if you want to update the links. Manual links are updated only when you select a menu command to do so.

The advantages of linking include smaller file sizes than with embedding, since all that exists in a linked document is the path to the source file. You can also make sure that multiple documents all contain updated information.

Disadvantages to linking include the fact that if you move the source file from the path specified in the linked documents, you must respecify the new path to make the links current. If the source file is deleted, the information embedded in the client document that is linked to that source file can no longer be updated, though a visual representation of the information remains in the document.

Packaging

With Windows operating system version 3.1, you can now embed objects from applications that aren’t specifically designed as OLE servers. You do this by using a program for Windows 3.1 called Object Packager (which is included in the Accessories Group in Program Manager). You create a *package* which consists of an icon (which you can customize) and a path to the server application and the correct file. That package is then embedded in the client document.

For example, you might have occasion to embed information from a spreadsheet that was designed to run on the MS-DOS but not the Windows operating system—Lotus® 1-2-3® for MS-DOS, for instance. You could use Object Packager to package the spreadsheet information and embed it in an electronic mail message or some other application, even though the spreadsheet information is not from an OLE-capable application.

While a package can originate from *any* application that runs in Windows, they can *only* be embedded in applications that have been specifically designed as OLE clients. Consult with your software vendor or read your software documentation to determine if your applications are designed as OLE clients.

A package behaves just like any other embedded object. When you double-click the package icon, the Windows operating system starts the server application, which can display a chart, play a recording, or perform whatever function the application provides. You can both view the object and edit it.

How to Use OLE

With OLE, you can build a compound document in four ways:

1. **Embed information** from another OLE application in the document. You see the object in the new document exactly as it appears in its original application. The embedded object includes data that allows the server application to launch from within the client application so that the embedded object can be edited.
2. **Set up a link** between your document and a file created in another OLE application. You see the object as it appears in its original application, but the link doesn't allow you to launch the server application. The source file must be opened in the server application and then edited. All links to the source file are automatically updated.
3. **Link an embedded object**. With this feature, you can maintain a link to the source document in the compound document as described above and also launch the server application from within the client application.
4. **Embed or link information from a non-OLE application**. An icon appears in the compound document to represent the object. This is done through Object Packager.

Here are the general procedures for embedding and linking according to the OLE specification.

To embed an object

There are two different ways to embed an object. You can start the process in either the compound document where you are working, or in the server application where you want to create the object. For example, you are using a word processor to create a document called REPORT.DOC. While you're working on REPORT.DOC, you decide you want to add information on profits for the month. The best way to create this information is in a spreadsheet program. You open the spreadsheet program, create the spreadsheet, highlight the information you want, and embed it into the report. (Please note that the user interface for embedding depends on the client application.) This information can then be edited later on. The spreadsheet does not have to be saved as a separate file, since it all the information needed to access the spreadsheet is contained in the host document.

Some applications also allow you to create an embedded object from within the client application.

To link an object

While you are analyzing construction costs for the new factory that you are describing in REPORT.DOC, you decide to link parts of a monthly expense report that is itemized in a spreadsheet. Instead of creating an embedded object, you decide to link the spreadsheet that already contains the information you need to your report. By linking the two documents, you know that every month your report will be automatically updated when the spreadsheet is updated.

The link is represented visually in the client document by the part of the spreadsheet that you selected. However, because it is only linked, you have to open the source file in the application in which it was created in order to edit the information. Once the edits are completed and saved, all the links to the source file are automatically updated.

You can also do a link to an embedded object, depending on whether the application you are using supports this feature. With a link to an embedded object, you can access the server application from within the client application while maintaining a link to the original document.

Additionally, you can package a document or piece of information using Object Packager. The package itself is embedded, but the information contained in the package can be linked to a source document. Double-clicking the icon that represents the package in the client document launches the server application and loads the information contained in the package. Packages are particularly useful for including whole documents inside other documents, particularly for providing further information on a topic, and for inserting MS-DOS–based command lines.

Please note that you can only insert a package into an application that supports object linking and embedding.

Additional ways to use OLE

The following scenarios illustrate more specifically some of the ways you can use OLE.

Embed a diagram.

It's your job to prepare the monthly status report on the new factory construction project. It would be easier to explain what's happening if you could refer to an up-to-date diagram of the factory.

You create the diagram in your favorite drawing program and embed it in your report. Each month, when it's time to prepare the next report, you just double-click the drawing. The drawing program starts with the factory layout loaded, so you can update the diagram. When you quit the drawing program, you're back in your report document with the updated diagram in place.

Link to a spreadsheet.

Some of the people who receive your factory status report need up-to-date cost information, sometimes week by week or day by day. The accounting department maintains the cost information in a large spreadsheet.

You add an object to your report that is linked to the summary cells in the accounting department's spreadsheet. Now you distribute the online version of the status report by electronic mail. Whenever someone opens the online report, they automatically have the latest summary figures. If they need more detailed cost information, they can double-click the summary information to open the complete spreadsheet.

Package video sequences.

You're writing a procedure manual for your factory workers that will be available online at workstations that have multimedia capabilities. Most of the manual consists of brief procedures and checklists written as job aids for experienced workers. But your inexperienced workers need more detailed instruction.

You use Object Packager (found in the Accessories Group in the Windows 3.1 Program Manager) to embed video sequences that show detailed instructions for various procedures. When a worker needs to know more about a procedure, two mouse clicks on an icon start the video player program and play the video right on the computer monitor.

Package your own application.

Your transportation scheduling program—an application you developed within your company—plans the routes your trucks will take each day. The supervisors at your warehouses need that information from time to time.

You link the output from that program to your daily order status

report, which you maintain in Microsoft Excel and distribute over your network. Now when the supervisors need to see the transportation schedule, it takes just two clicks on an icon to display it.

Creating a compound document with OLE

The following illustration shows an example of an OLE compound document. It's a draft of a report on a proposed new factory that the author is putting together to hand over to the graphic design team. In addition to the text of the report, the author also wants to use:

- Part of a spreadsheet and graph that show financial analysis.
- Scanned images of several photos.
- Slides from a presentation.
- An artist's rendering of the factory that's under construction.
- A floor plan of the factory.
- A bitmap of the company's new logo.
- Recorded instructions to the graphic designers.

This is a good example of application integration: The author knows what kinds of information needs to be included in the document and can use the best program available to create each type of information.

µ §

The two-headed arrows in the diagram illustrate that each of the objects —whether linked or embedded—**is a dynamic, changeable element**. Once the author has assembled these objects in the compound document, he can double-click any of them and OLE will open the application that created the object so that it can be edited. When the author exits the server application, OLE automatically returns to the compound document.

How OLE Works

In OLE, information is exchanged between applications as objects. Objects are said to be "opaque" to the client application, because the client doesn't access the object and doesn't need to understand the data it contains. Instead, the client application simply calls application programming interfaces (APIs) in the OLE dynamic-link library (DLL) whenever it needs to display an object or invoke the object's server application.

µ §

Objects are stored as part of the compound document file. An embedded object contains the native data from the server application and the name of the application. A linked object contains the name of the server application and the complete path to the server file. (If you move the linked file or change its name, the link is broken, but you can easily reestablish the link.)

Both kinds of objects can also contain presentation information such as a metafile image of the source information. This allows the client application to display objects without having to invoke the server applications.

The server and client applications and the OLE libraries must all be on the same computer. Linked objects, however, need not be on the same computer. In fact, keeping the source file on a network file server and linking to it from several documents is an excellent way to make sure that users in different locations all have the most current data.

µ §

When server applications are installed on the computer they register themselves in the system registration database (REG.DAT). When a client activates a linked or embedded object, the client library finds the listing for the server application in the database and uses the information there to start that application server. Client applications refer to REG.DAT for the list of OLE server applications to display in the Insert Object dialog box and for the name of the server application for the Paste Special dialog box.

Appendix A: Common Questions and Answers about Object Linking and Embedding

Q: Can you establish an OLE link to data on a mainframe?

A: Yes, you can establish a link to the raw data so that when the data on the mainframe changes, it changes on your PC in the application that you use to receive the data. You can use a server application to capture the mainframe data and package it, and then embed it in another client application where you create reports. Once the link is established, a change in the mainframe data will trigger an automatic change in the spreadsheet and, subsequently, in the document.

Q: What happens when you send an OLE document to a user who doesn't have the server application on his or her machine?

A: The person who doesn't have the server application will still be able to see the presentation of the OLE object. However, they won't be able to edit it. If they double-click the presentation, an error message will appear telling them that the server application cannot be found. (If the second user passes the document on to a third user who *does* have the server application, the third user will be able to edit the linked or embedded object.)

Q: Would you ever need to use DDE instead of OLE?

A: OLE is a superset of DDE, so there may be instances where a DDE link is preferable. Suppose you want to bring some data from a spreadsheet into a word processing document, but you want to format the data in the word processor instead of presenting it in its spreadsheet form. If you embed the data, you can't format it in the word processor; but, if you link it, you can also format it.

Q: When would you use Paste Link or Update Link, as in Windows environment 3.0?

A: Paste Link is still the command to use in the client document to add a linked object. By default, the link you create will be an automatic link. If you change the link to a manual link, then you will need to use the Update Link command whenever you want to update the information in the client document.

Q: If my document contains a link to a file on a network server, will OLE establish the network connection when I double-click the object?

A: Yes. If the network server is password-protected, Windows will ask you to enter the password. If the net server cannot be located or accessed, you will get a message that says “Link not found” along with a Change Link dialog box that will let you browse for the net connection. Then you can quickly rebuild the link so you can view the object.

Q: How are the dynamic-link libraries used?

A: Applications use three dynamic-link libraries, OLECLI.DLL, OLESVR.DLL and SHELL.DLL to implement object linking and embedding. Object linking and embedding is supported by OLECLI.DLL and OLESVR.DLL. The system registration database is supported by SHELL.DLL. These libraries may be included with any products developed for Windows version 3.1, to ensure that the products will run with Windows version 3.0.

Q: How does communication happen between OLE libraries?

A: Client applications use functions from the OLE API to inform the client library, OLECLI.DLL, that a user wants to perform an operation on an object. The client library uses DDE messages to communicate with the server library, OLESVR.DLL. The server library is responsible for starting and stopping the server application, directing the interaction with the server’s callback functions, and maintaining communication with the client library.

When a user modifies an embedded object in a server application, the server application notifies the server library of changes. The server library then notifies the client library, and the client library calls back to the client application, informing it that the changes have occurred. Typically, the client application then forces a repaint of the embedded object in the document file. If the user changes a linked object in the server application, the client library is notified that the object has changed and should be redrawn.

Appendix B: Features of OLE that DDE Does Not Support

OLE Feature	Description
Extensibility to future enhancements	The OLE libraries may be updated in future releases to support new data formats, link tracking, and *in situ* editing.
Persistent embedding and linking of objects	The OLE libraries do most of the work of activating objects when an embedded document is reopened, by reestablishing the conversation between a client and server. In contrast, establishing a DDE link is the responsibility of either the user (if the link is not persistent) or of the application (if the link is persistent).
Rendering of common data formats	The OLE libraries assume the burden of rendering common data formats in a display context. DDE applications must do this work themselves.
Server rendering of specialized data formats	The OLE libraries facilitate the rendering of specialized data formats in the client's display context. The server application or object handler actually performs the rendering. The client application has to do very little work to render the embedded or linked data in its display context.
Activating embedded and linked objects	The OLE libraries support activating a server to edit a linked or embedded object. Activating servers for data rendering and editing is beyond the scope of DDE.
Creating objects and links from the Clipboard	The OLE libraries do most of the work when an application is using the Windows Clipboard to link or exchange objects. In contrast, DDE applications must call the Windows Clipboard functions directly to perform such operations.
Creating objects and links from files	The OLE libraries provide direct support for using files to exchange data. No DDE protocol is defined for this purpose.

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This document is for informational purposes only. **MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT**.