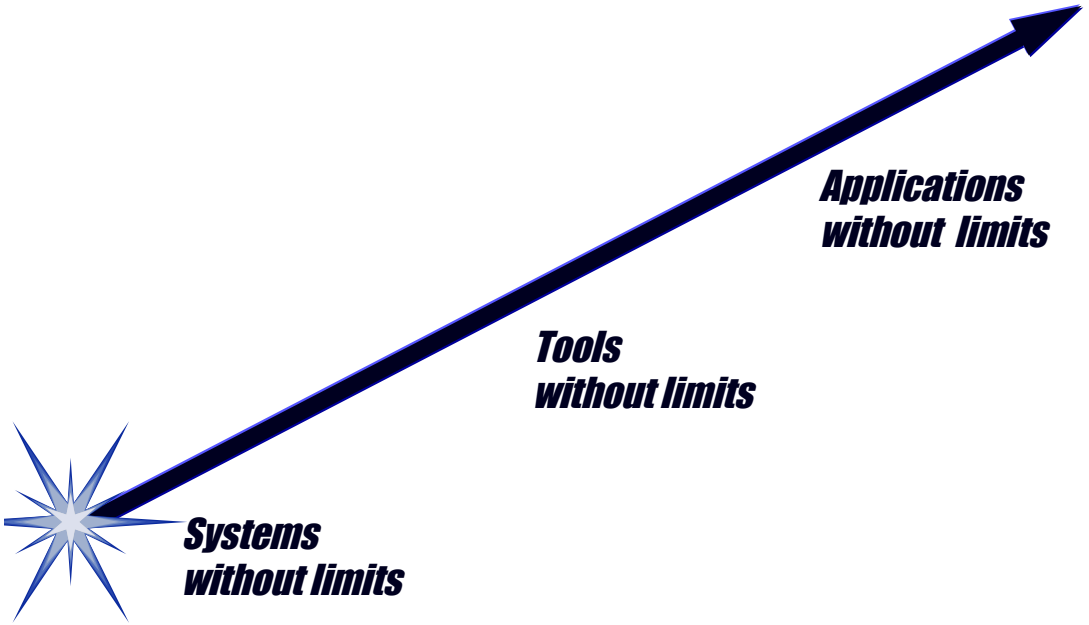
Corporate Backgrounder

# *Microsoft's Object Technology Strategy: Software Without Limits*




March, 1994

**(0394 Part No. 098-55163 )**

Microsoft®

## Contents

## *Executive Summary*

Computers must make it easy for corporations to access and use strategic information, regardless of its source or location. This goal forms the basis for Microsoft's vision of *Information at Your Fingertips,* and new innovations in object-oriented software technology are a significant step towards the realization of this vision.

Object technology will improve the ability of corporate information departments to deliver strategic information systems at lower costs, and will help businesses compete in a global economy. Microsoft, as a leader in object technology, is pioneering advances in application development environments, desktop applications, and system software. Corporations can receive the benefits of Microsoft object technology today through applications and development tools supporting Object Linking and Embedding (OLE), a breakthrough in object-enabling system software.

The primary benefits of object-oriented programming are directed at programmers. OLE, on the other hand, brings the benefits of object technology directly to end-users, while also addressing the needs of corporate developers and system integrators. OLE 2.0 is available today for the Microsoft Windows™ family of operating systems, and it will soon be available for the Apple® Macintosh® platform. In the future, software components based on OLE will also be able to interoperate across all major versions of UNIX®, VMS®, and even the MVS operating system. In fact, because OLE is based on an advanced underlying object architecture called the Component Object Model (COM), OLE forms the basis for Microsoft's strategy to evolve the Windows family into fully object-oriented operating systems. Independent software vendors, system integrators and corporate information departments can begin implementing solutions today using OLE, and be assured that any OLE 2.0-enabled application available today will transparently integrate with the distributed, object-based system services that will be incorporated into the Windows family of operating systems.

But the future begins today: OLE is the most advanced, cross-platform system object model available, and software components based on OLE are already revolutionizing the software industry.

***Microsoft's Strategy for Delivering Object Technology Innovation***

## *Overview*

Increasingly, businesses are using information system technology as a strategic tool to provide long-term competitive advantage. The growing role of information systems as an integral part of corporate strategy has been enabled by advances in computing technology, and has been driven by the dynamic nature of a global economy. As corporations adjust to increased competitive pressures, they have turned to information technology not only to reduce costs and improve workflow management, but also to deliver better products and service to their customers. At the same time, advances in personal computer hardware and software technology, such as client-server computing, have enabled information systems to meet these new needs.

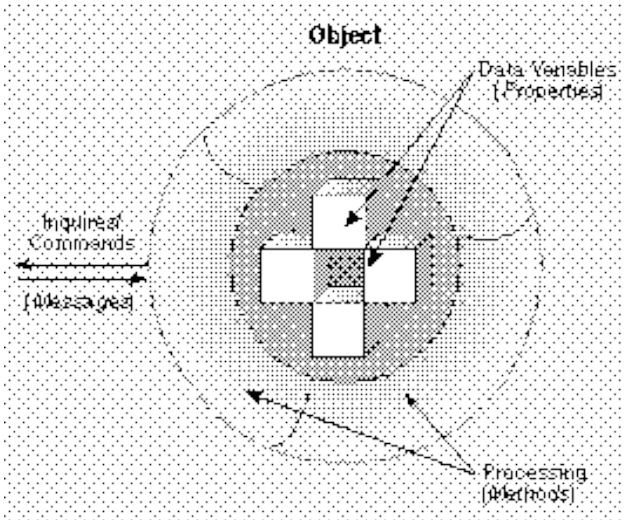
The strategic use of information technology, however, has also placed a growing burden on corporate information departments. These departments are now asked to deploy better systems at a faster pace and, at the same time, keep up with rapid advances in software technology. While various development tools have helped ease the burden to some extent, most corporate information departments still face a growing application backlog, and deal on a day-to-day basis with complex programming and system deployment issues. *Object Technology,* however, has the potential to offer substantial benefits to information departments as they deploy new and innovative applications. These applications will, in turn, help corporations meet the challenges of a dynamic, competitive economy.

### Object technology: the future begins now

In developing computer software, object technology offers a different model from traditional structured programming and design, which is based on functions and procedures. In simplified terms, object technology is a way to develop software by building self-contained modules that can easily be replaced, modified and reused. Object technology will empower information departments to deliver more flexible, higher quality systems, and it will reduce the time and resources required for system programming, implementation and maintenance.

Imagine building an innovative billing system to handle new international subsidiaries by assembling 75 percent of the new application from existing components, and delivering that system in a fraction of the time it would ordinarily have taken to develop. Imagine being able to integrate this billing system quickly and easily with global network-based information resources such as database and messaging services, even though these systems may reside on a variety of different hardware and software platforms. And imagine extending the system to incorporate charting and statistical sales analysis by simply buying packaged products and plugging them into the new system. Although this might sound unattainable, Microsoft's object-enabling system software*, OLE,* is making all of this possible.

***Anatomy of an Object***




*An object is a self-contained software module that encapsulates both data and processing details.*

***An Object-oriented Billing System***

An object is a self-contained software module that consists of a set of data and its associated processing information. A significant benefit of object technology is that an object *encapsulates* all of the data and processing details, hiding its inner complexity from programmers and users. This makes it easy to use objects once they are defined. Objects are also more easily protected from misuse, since they can only be accessed through well defined interfaces, called *methods*. In addition, since all implementation detail is hidden from other objects (other software modules), it is easy to modify an object's internal details without affecting any other objects in the system. As a result, object-based systems are much more flexible and easier to maintain than their procedural-based counterparts.

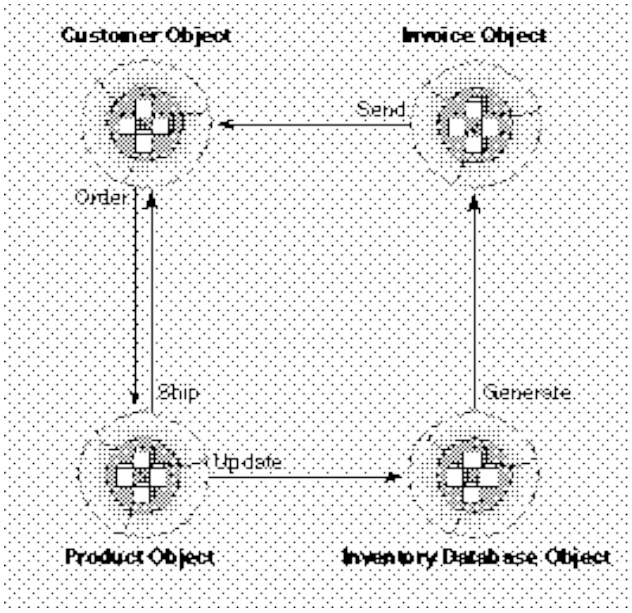
## Object-enabling system software vs. object-oriented programming

The two areas of object technology that will have a growing impact on business systems are the following:

- **Object-enabling system software**
- **Object-oriented programming languages**

The issues and goals of object technology differ for the two areas, although some of the basic concepts are the same. In fact, the term object technology is hard to precisely define because it is so often applied to each area without taking the differences into account. Object-oriented programming languages and development tools are useful for building self-contained, custom applications by creating object definitions in the form of source code. These language-based object definitions can be shared and reused in different applications. Object-oriented programming languages, however, do not provide a means for separate applications to be integrated with other custom applications or packaged software. They do not fulfill the need for diverse objects, supplied by any company, written in any programming language, to freely interact. To fulfill this critical need, object-enabling technology must also be incorporated into system software, and applications must be designed to use these system software capabilities.

*Objects hide their inner details from programmers and users. As a result, programmers and users can be more productive, since they can use objects without learning their complexities. Objects can also be easily modified or replaced without affecting other objects in the system.*



## *Object-enabling System Software*

The role of system software in object technology innovations is critical, and the issues are complex. While object-oriented programming can be used to build individual applications that run as a single process, system software must provide a bridge between diverse applications, addressing the following critical needs:

- The need to smoothly integrate objects written by different companies using different programming languages
- The need for an overall object model to facilitate object communication across application and machine boundaries (i.e. across a network)
- The need to enhance and upgrade objects autonomously, without disrupting the operation of a distributed system

As organizations optimize and re-engineer their business processes, they need to be able to fully integrate a variety of information, applications and systems in a flexible, work-flow driven manner. An object-oriented system environment makes this possible. In such an environment, virtually everything is accomplished by a large number of objects that interact with each other to perform work. Even a simple operation such as dragging a work form out of an application and onto a printer icon or message outbox can involve dozens of objects communicating with one another. With many objects that need to cooperate with each other, it is important to have a well-designed object model that formally defines what objects are and also defines the rules of object interaction. This is the role of object-enabling system software, since it defines the way autonomous objects interact with the outside world, not just the way object classes within a given program are implemented.

If designed properly, object-enabling system software makes objects useable and reuseable across application boundaries. The object-enabling system software ensures that objects written by different programmers from different companies behave in a well-known and consistent manner--without constricting how programmers implement different objects. In more technical terms, the object-enabling system software defines a binary object interface that is independent of programming language. Such a model also defines a mechanism to ensure that connections between objects are valid, even as objects in a distributed system are individually upgraded or replaced. Object-oriented programming languages, on the other hand, define a programming language (source code) standard, making their objects dependent on both language and implementation. These languages assume that when an object is upgraded, dependent applications can be recompiled and re-distributed simultaneously. This is unrealistic in a distributed system, where components are typically supplied by different companies and different programming teams. Unfortunately, some object systems (such as IBM SOM/DSOM, and the Apple OpenDoc™ specification) attempt to straddle the border between a true system object model and object-oriented programming technology. The result is an object system that offers some benefits, but lacks the robust, binary object standard and version management that is necessary to achieve true application integration using components from multiple vendors.

To say that the advantages of a robust, system software object model are significant is to understate their importance. With such object-enabling system software:

- Users can manipulate objects that represent *any* information including text, graphics, reports, and even multimedia clips, across application boundaries, no matter who designed the object or what language or development tool was used to program the object.

- Through a standard programmatic interface based on the system object standard, off-the-shelf, packaged objects can communicate with each other and be integrated into complete line-of-business solutions. This capability is called *component software*. Component software offers a more efficient and productive model for the software industry, and will dramatically reduce the programming effort and time required to deliver new business solutions.

Increasingly, corporations, system integrators and independent software vendors are discovering that object-oriented development languages and tools can offer programming benefits, but without robust, object-enabling system software, the promises of object technology will largely remain unrealized. This is not to say that object-oriented programming languages are not beneficial. The scope of issues these languages attempt to address (ease of development effort through maximum code reuse), however, is different than the scope of issues addressed by object-enabling system software. In fact, developers can use these two types of object technology together, using object-oriented programming languages to build component objects based on the system object model.

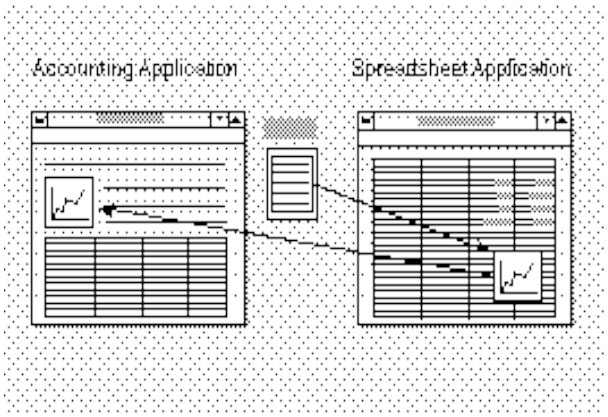
In the following pages, you will see how Microsoft, working closely with other industry leaders, has designed object-enabling system software that is already beginning to revolutionize the software industry, bringing the benefits of object technology not only to programmers, but also directly to end users.

## Object Linking and Embedding 2.0 and the Component Object Model

OLE 2.0 is a breakthrough in object-enabling system software. OLE is based on the *Component Object Model* (COM), an underlying system software object model that allows complete interoperability between objects that are written by different companies and/or in different programming languages. These objects can be purchased, replaced, enhanced and reused at any time during the business-system life cycle. The primary responsibility of the Component Object Model is to ensure that objects behave in a well-known and consistent manner without constricting how programmers implement different objects. The Component Object Model accomplishes this by defining a binary interface for objects that is independent of any programming language. Objects conforming to the Component Object Model can communicate with each other without being programmed with specific information about each other's implementations.

Objects that are written to support the Component Object Model are collectively called *component objects*. Every feature of OLE depends on the Component Object Model to provide basic inter-object communication. In other words, the Component Object Model provides the "plumbing and wiring" of OLE. Using OLE, for example, a spreadsheet object provided by one vendor can be seamlessly embedded into a word processing document created by an application from another vendor. The

***OLE Automation***




*Pictured above, an accounting application uses OLE Automation to send commands to a packaged spreadsheet application to update a table of sales figures. It then uses the spreadsheet application to create a chart and automatically insert the chart object into a quarterly report.*

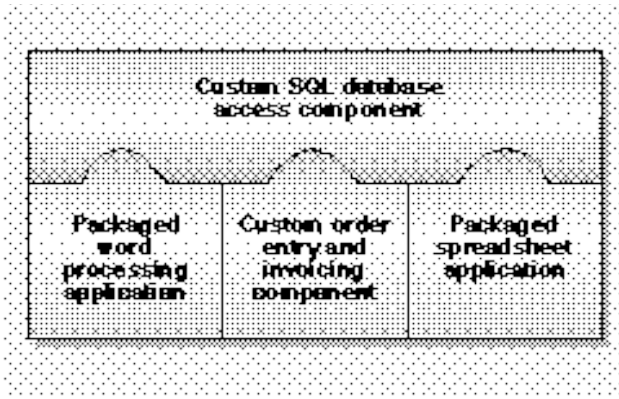
spreadsheet and word processor don't need to know anything about each other's implementation, they only need to know how to connect through the interface provided by OLE.

OLE is a set of object services built on top of the Component Object Model. While many of these services are related to *compound documents,* OLE is much more than a compound document architecture. OLE provides a robust platform for building custom business applications that can be easily integrated with other business applications as well as with packaged software, whether the applications execute on a single machine or are distributed across a network.

## Component software

OLE *Automation* allows an application to take advantage of services provided by other OLE-enabled applications. These services can vary widely from application to application, but they are provided through a standard object interface. The interface is used by software vendors to expose any application-specific capability through a cross-application, object-based interface. In this way, corporate developers and system integrators use traditional programming languages and development tools to write applications that access these capabilities. Today, such tools include Microsoft Visual Basic, Microsoft Visual C++, and many third-party development tools as well. Even more development tools will soon be supporting OLE Automation programming capability.

As an example, a custom accounting application can use OLE 2.0 Automation to activate a packaged spreadsheet application, fill in and format spreadsheet cells, then use the spreadsheet application charting engine to create a graph for a quarterly report. Through OLE Automation, the graph can be automatically embedded into the quarterly report before the report is printed for distribution.




***Component Software***

As this example shows, corporate developers and system integrators can use OLE Automation to quickly assemble larger, custom business solutions using packaged, component software as building blocks. OLE Automation allows developers to easily integrate custom software with any component software, including productivity applications, packaged vertical-market applications, and any OLE-enabled application or development tool. In short, OLE breaks down the barriers between different types of software and allows any packaged or custom application to be programmed to communicate with other applications to achieve a business solution.

## OLE Custom Controls

*Through OLE, packaged applications can be integrated with custom software to form complete solutions. The use of component software will significantly reduce the programming effort required to deliver new systems, while increasing system quality and flexibility.*

Today, Microsoft Visual Basic provides a specific class of objects called *Visual Basic custom controls* (VBX's). Visual Basic custom controls are an example of the benefits of component software. Hundreds of Visual Basic custom controls are commercially available, providing a wide range of services such as report writing, data access and messaging services. These controls can be obtained from many different companies and easily incorporated into any application written in Visual Basic. While the VBX architecture has been rapidly accepted by developers, it was not designed to be an open, standard interface.

The VBX architecture is closely tied to the Visual Basic design environment, making it difficult for existing Visual Basic custom controls to work with other development tools, applications and objects. Thus, a particular Visual Basic custom control cannot be used on multiple hardware platforms, multiple operating system platforms, or in multiple development environments.

*OLE Custom Controls*, however, do not have these shortcomings. By merging the benefits of OLE with the existing VBX architecture, Microsoft is providing corporate developers, system integrators and independent software vendors an open, standard way to receive the benefits of component-based software development. As an extension to the existing OLE Automation interface, the OLE control architecture will allow OLE Custom Controls to work with today's OLE-enabled applications across multiple development environments, and across different hardware platforms and operating systems. For example, users will be able to insert different OLE Custom Controls into a database development environment to provide a range of capabilities for their database applications. These capabilities might include specialized financial modules, equation editing, scientific analysis, run-time tutorials, messaging, or any other type of capability. The same OLE Custom Controls will also work with other development tools, such as fourth generation programming languages. Users will even be able to incorporate the controls directly into any OLE-enabled application, including productivity applications such as spreadsheets and word processors. Because they will be able to pick and choose from a wide variety of standard, interchangeable components, users get exactly the functionality they need, in a more cost-effective manner.

Through OLE Automation and OLE Custom Controls, component software will help lower costs for IS departments. Corporations can now use packaged OLE components to reduce the amount of programming needed to deliver a custom solution. OLE Custom Controls are a quick, simple, efficient and standard way to incorporate reusable software components into custom and packaged applications. OLE Automation is a way to integrate packaged applications into complete custom business solutions. Additionally, solutions based on OLE component software can be easily adapted and extended by simply plugging in new components.

### Other features of OLE

In addition to Automation and the Custom Control architecture, OLE provides the following:

- **Object linking**
- **Object embedding**
- **Object storage**
- **Object Visual Editing**
- **Object drag-and-drop capability between applications**

Through *object linking*, applications can be linked to data objects within other applications. For instance, a spreadsheet table can be linked into multiple custom business reports, and as changes are made to this table within the spreadsheet application, all report documents are automatically updated. *Object embedding* is the ability to embed an object within another document without maintaining a link to the object's data source. In both object linking and object embedding, applications supplying objects are called OLE *servers*, while applications containing objects are called OLE *containers*. An application can be both an OLE container and an OLE server. OLE objects can be nested in multiple layers within other linked and/or embedded objects. Through *object storage*, the OLE system software allows applications to store embedded and/or linked objects in their native file structures, creating a persistent link between the objects and the container applications.

*Visual Editing* allows users to create rich, compound documents easily, incorporating text, graphics, sound, video and other diverse object types. Instead of switching between applications to create parts of the compound document, users can work within the context of their document. As the user begins to edit an object that originated in another application, such as a spreadsheet or graphic, the menus and tools of the container application automatically change to the menu and tools of that object's native application. The user can then edit the object in the context of the document, without worrying about activating and switching to another application. Additionally, through OLE *drag-and-drop* capability, users can select objects from an OLE-enabled application and drag them into other OLE-enabled applications. This eliminates the need to cut and paste and saves time by making data exchange graphical and intuitive.

## *Distributed Object Systems*

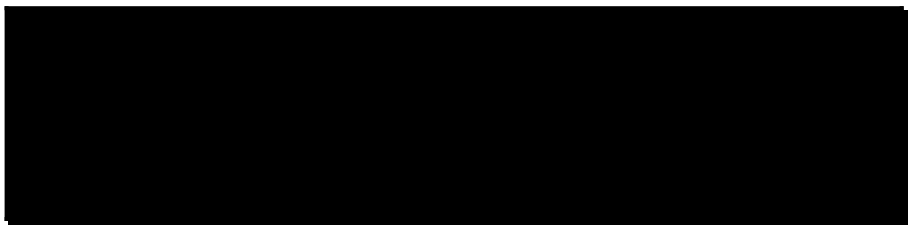
A key feature of the Component Object Model is its ability to facilitate communication between objects across a network. Today, OLE with distributed object support has been distributed to almost 5,000 developers in pre-release form. All OLE 2.0-enabled applications that are available today will be able to take advantage of distributed object capability in the future without any modifications. This capability will allow the seamless interoperation of OLE-enabled applications running on different machines, using a standards-based *Remote Procedure Call* (RPC) mechanism. The RPC mechanism, called Microsoft RPC, is based on and compatible with the Open Software Foundation's Distributed Computing Environment RPC (DCE RPC).

## Example: a real-time stock-quote object

To illustrate the power of OLE with distributed object support, imagine that a user wants to create a spreadsheet that includes a link to a real-time graph of stock prices. With a packaged OLE-enabled stock-quote object running on a centralized server, the user can simply choose "insert" from the spreadsheet menu, pick the stock-quote object from a list of available objects, and link the stock-quote object into the spreadsheet running on the desktop. Through OLE's distributed object support, the user receives a real-time stock graph in the spreadsheet, but the stock-quote object itself, with its live data feed and associated data processing, is executed on the server across the network.

In addition, because the stock-quote object is a standard COM object supporting OLE linking, it can supply live data to any OLE-enabled container application without any additional programming. Thus, it could just as easily be placed in a word processing document, or a custom financial trading application. To extend the example even further, the stock-quote object might incorporate local OLE Custom Controls that include a toolbar to allow individual users to change the types of stocks that are visible on the real-time graph. As individual users move the object into specific documents or application windows, they could easily access the toolbar simply by clicking the object. Then, they could use the toolbar to change which stocks are shown on the desktop. No additional code is needed to allow the object to be reused and customized for these individual uses. The true power of a robust system software object model is that it brings object use beyond the realm of programmers sharing source code. It allows users *and* developers to manipulate, customize and reuse objects without any programming whatsoever.

OLE with distributed object support will allow corporate developers to split an application into component modules that can each transparently execute on a different computer. Since the Component Object Model will provide network transparency, these components appear to users and programmers to be located on a single machine, as illustrated in the previous example. The object-oriented billing system described earlier could be designed as a set of components, including: a database query component; an invoice processing component; a forms builder; and a transactions manager. Each of these components could run on a computer suited to the amount of processing power and disk capacity the particular component requires. In fact, the system could be designed to dynamically locate idle processing power available on the network and distribute the individual components to specific computers accordingly.



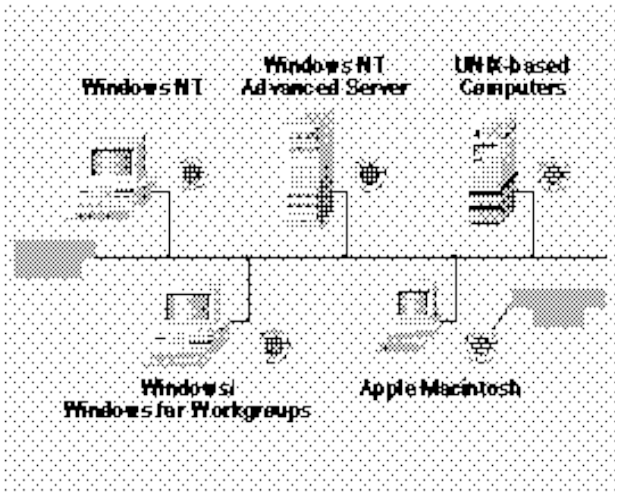
Furthermore, distributed object technology can also be used to make global networks and information resources appear to be local, making it easier and faster for users to access critical business information. Through OLE with distributed capabilities, users will be able to locate and execute objects across global networks, without even knowing that the information is thousands of miles away. The distributed

capabilities for OLE will be incorporated into future versions of the Microsoft Windows family of operating systems. Because OLE is an open, cross-platform technology, corporations will also be able to assemble distributed OLE solutions across heterogeneous computing environments.

## OLE 2.0 and cross-platform capabilities

Microsoft continues to work with other industry members to help ensure that object technology advances within the Windows operating system are available on many diverse platforms, and can interoperate with different systems. For example, Microsoft is providing OLE 2.0 capability for the Apple Macintosh platform by layering the technology on the System 7 operating system. This will allow cross-platform support for compound documents, allowing them to be freely exchanged between Windows and the Macintosh. Additionally, Microsoft will make remote OLE Automation and data transfer available between OLE-enabled applications on the Macintosh and those running on the Windows platform.

***OLE With Distributed Capabilities***




*All existing OLE 2.0-enabled applications will be able to take advantage of future distributed capabilities without modification. Furthermore, OLE will be available on a wide range of platforms.*

Furthermore, through a recent agreement with Digital Equipment Corporation, Microsoft and Digital are integrating the Component Object Model technology with Digital's ObjectBroker™ to enable cross-platform OLE capabilities. Through ObjectBroker, OLE-enabled applications running on Windows or the Macintosh will be able to use and interoperate with objects running on SunOS™, IBM AIX®, HP-UX®, DEC ULTRIX®, OSF/1 and OpenVMS through OLE data transfer and OLE Automation. Already, Microsoft and Digital have demonstrated a Windows-based Microsoft Excel spreadsheet containing a linked stock quote object that runs on an OSF/1 server. As with the example discussed earlier, the stock-quote object itself executes on a centralized server (the OSF/1 machine). Because it is a standard COM object supporting OLE 2.0, it can also be freely linked with any other OLE-enabled desktop application. In addition to the services already defined in OLE 2.0, corporate developers and ISVs will be able to create many new types of objects and interfaces that are both source-code portable across platforms, and fully interoperable across a network.

## Other object model specifications

The object-enabling system software arena can be confusing because there are different object models that have been proposed by various vendors. For any system software object technology, even those that claim to be cross-platform and interoperable, several key questions should be addressed. These questions include the following:

- Is the object model a proven technology with many available applications, or is it merely a paper specification?

- Does the object model address the need for objects to work seamlessly between applications on the same machine, and can the objects also work seamlessly between machines running over a network using exactly the same model?
- Does the object model provide a safe binding mechanism? In other words, are there cases in which objects could connect and fail because of weaknesses in the object system?
- Does the object model address the need for open, cross-platform, interoperability?
- Does the object model support a secure environment for distributed applications?
- Will the object model allow seamless integration of application objects and objects exposed as operating system services? That is, is the object model appropriate for use in extending operating system services, providing seamless integration between the operating system and applications?

OLE and the Component Object Model are open, proven technologies that meet all of these needs. OLE is by far the most advanced object-enabling system software available, and it offers new potential for personal as well as corporate computing. Object models such as IBM SOM/DSOM, and the Apple OpenDoc specification for compound documents do not address many of these areas. Please refer to the accompanying insert for a more detailed, technical comparison of these specifications.

### The Open Process

OLE 2.0 was refined through an open forum Microsoft calls the *Open Process*. Through the Open Process, Microsoft freely distributes and discusses preliminary technology specifications with hardware and software vendors, software architects, OEMs, and corporate developers a year or more before the scheduled release. They, in turn, offer alternative approaches and solutions that help refine and shape the system technology. Early access to the design process also allows independent developers to build applications that fully exploit the new technology, giving users more innovative and robust applications.

Major software vendors, including Lotus Development Corporation, Borland International, Inc., Apple, Computer, Inc., WordPerfect Corporation and many others participated in open design reviews for OLE 2.0. In addition, preliminary OLE 2.0 specifications were distributed to more than 150 other software vendors for review and feedback. As a fully published and open specification, OLE 2.0 has enjoyed tremendous industry acceptance. Hundreds of applications that support OLE 2.0 are either shipping or soon to be released, and OLE 2.0 won the 1993 *PC Magazine* **Technical Excellence Award**, the **MVP: Software Innovation Award** from *PC-Computing* and the **Technology Award for Excellence** from *BYTE Magazine.*

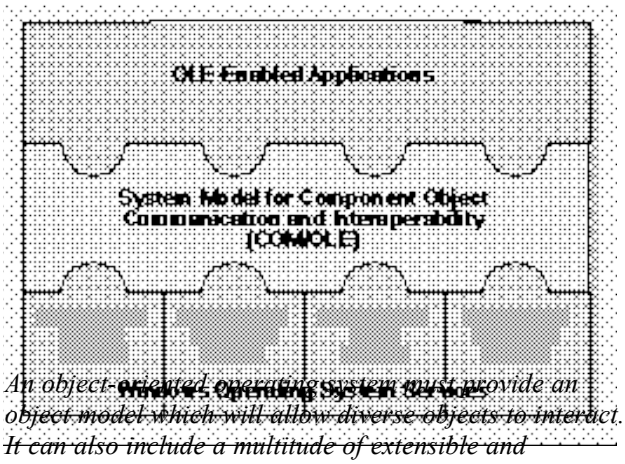
## *The Evolution of Microsoft Windows*

### Exposing system services as objects

As evidence of its advanced design, the Component Object Model not only allows individual applications to interoperate and communicate through OLE, it also provides an object-based interface to Windows system services, called system objects. Today, these services include memory allocation, file management and data transfer. In fact, OLE forms the basis for Microsoft's strategy to evolve the Windows family into object-oriented operating systems. Future versions of Windows will extend object-based services to include facilities such as integrated OLE Custom Controls, multimedia services, data-access services, name services, and distributed security. These object-based services will gradually become pervasive, providing easily replaceable components within the Windows operating system itself.

***OLE: Providing Tight Integration Between Applications and Windows System Services***

This evolution will protect a corporation's investments in existing software, since current Windows-based applications will continue to operate on future versions of Windows. The OLE interface will become the single standard interface not only for applications to interact with each other, but also for applications to interact with operating system services. As the object capabilities of Windows expand, existing applications will get many new object-based features for free. All existing OLE 2.0 applications, for instance, will be able to work seamlessly across a network through OLE with distributed capabilities. And existing OLE-enabled applications will support extensive drag-and-drop capabilities within the Windows user-interface, as discussed below.




*An object-oriented operating system must provide an object model which will allow diverse objects to interact. It can also include a multitude of extensible and replaceable component services, exposed to applications as objects.*

Object-based innovations that will be incorporated into future versions of Windows include the following:

- **An improved object-based graphical user interface**
- **Diverse system services exposed as system objects through OLE-style interfaces**
- **Integrated OLE Custom Controls**
- **A new object-based local and distributed file system**

The next version of Microsoft Windows (a major upgrade from Windows 3.1, code-named "Chicago"), as well a future release of the Windows NT™ operating system (code-named "Cairo"), will incorporate an object-based user-interface. This interface will treat all files and applications as objects, unifying the Program Manager and File Manager. The interface will allow users to manipulate files, programs, printers, utilities and other system resources in a consistent fashion. Users will be able to group logical combinations of objects into folders, and place frequently used objects on the desktop for easy access.

In addition, through extensive OLE support, OLE-enabled applications will be seamlessly integrated with the Windows user-interface, supporting drag-and-drop across the desktop and between system resources.

Furthermore, objects such as files and system resources will have a new feature called *property sheets,* which will present identifying information. This information will allow users to browse and configure objects without actually opening them or having to use special system utilities.

As it evolves, the next generation of Microsoft Windows NT will incorporate all of these advances, plus additional, advanced capabilities for high-end workstations (Windows NT) and server platforms (Windows NT™ Advanced Server). Windows NT "Cairo" will incorporate an object file system that will allow for very easy and exceptionally fast retrieval of information. Through object cataloging and properties a user will be able to search an entire network for all documents meeting a specified criteria. For example, a user could ask for all documents written by John Adams that contain 1994 budgeting information. Of course, there will continue be full security to protect files and resources from unauthorized access.

## Microsoft's strategy: pioneering object technology innovations

Microsoft is a leader in object technology and has pioneered new innovations both at the application programming level and the system software level. These innovations are a step toward realizing a vision we call *Information at Your Fingertips,* and they are aimed at bringing the benefits of object technology directly to end users. Computers must make it easy for corporations to access and use strategic information, regardless of its source or location. While Microsoft's work with object technology has brought us closer to the vision, many challenges remain. Microsoft will continue to be a leader in delivering object technology by operating according to the following strategic principles:

***Microsoft's Strategy for Delivering Object Technology Innovation***

## *Object Technology and Microsoft Products*

Today, Microsoft offers the benefits of object technology to enterprise computing through the OLE object-enabling system software, desktop applications that make full use of OLE capabilities, and object-oriented programming languages and development environments with integrated OLE support.

### Object-enabling system software innovations

- *Object Linking and Embedding 2.0*-- as stated by Michael J. Miller, Editor In Chief of *PC Magazine*, "OLE 2.0 offers a far easier, far better method of integration, and it will fundamentally change our expectations for the next generation of software." (*PC Magazine*, Dec. 7, 1993 p. 78). Many OLE 2.0-enabled applications are available today and hundreds more are on the way.

- *The Component Object Model*-- the object model on which OLE 2.0 is based, which will be an integral part of future object-oriented versions of Microsoft Windows. The future of Windows includes the ability to easily build distributed applications that interoperate through OLE, across different machines and different platforms. Today, the Component Object Model offers the ability to use OLE applications as off-the-shelf components for custom business solutions, and to program custom, OLE-enabled business objects.

### Desktop applications that make full use of OLE capabilities

- *The Microsoft Office Business Productivity Applications*-- includes the Microsoft® Word 6.0 word processor, the Microsoft Excel 5.0 spreadsheet, the Microsoft PowerPoint® 4.0 presentation graphics program, and the Microsoft Access® 2.0 relational database development system. These applications are OLE 2.0-enabled, with drag-and-drop capability, Automation, and Visual Editing. In addition, Visual Basic for Applications, a built-in macro language, can be used to customize Microsoft Office applications and integrate them with other OLE-enabled applications through Automation. Most other Microsoft applications will include OLE 2.0 support in upcoming releases.

### Object-oriented programming languages and development environments

- *Microsoft Visual C++*-- a complete visual development environment for the most popular object-oriented programming language in use today. Visual C++ can be used to build complete OLE-enabled applications. Through Object Wizards, developers can create OLE-enabled Windows applications at the touch of a button.

- *Microsoft Foundation Classes (MFC)*-- an application framework that provides hundreds of highly reuseable C++ object classes. These classes can be used to significantly reduce the amount of programming necessary to develop Windows-based applications, and to deliver complete OLE capability within these applications. MFC is bundled with Microsoft Visual C++ and many third-party C++ compilers.

- *Microsoft Visual Basic*-- a high-level, object-based version of the BASIC language, combined with a visual development environment that makes it easy to write custom, Windows-based business solutions. Today, Visual Basic provides a specific class of object called Visual Basic custom controls (VBX's), which can be used to add a wide range of services to a custom application. By taking advantage of the built-in support for OLE Automation, Visual Basic can also be used to customize and integrate packaged software. In the near future, OLE Custom Controls will also be available to bring the advantages of OLE to the existing VBX custom control architecture.

### Object technology: continuing innovation

Microsoft continues to refine its programming environments, applications and system software to deliver more functionality and increased ease of use. In the future Microsoft will continue to:

- Deliver best-of-breed development environments, applications and operating systems by listening closely to the needs of our customers, and by developing new object technology innovations that help meet these needs.

- Work closely with standards bodies and industry members through the Open Process to ensure interoperability and availability of Microsoft object technologies on a wide range of platforms.

## *Conclusion*

Today, advances in personal computing are helping corporations compete in a dynamic, global economy. Businesses, however, need new ways to deliver information systems in a faster, more cost-effective manner. Object technology, in the form of OLE, will help businesses meet these needs. OLE 2.0 is based on the Component Object Model, an object standard that allows objects written by any company in any programming language to interact. OLE addresses a critical system software requirement that object-oriented programming languages do not: the need for true application interoperability-- on a single computer and across computers operating on the network.

OLE will allow corporations to use packaged, component software as building blocks for complete business solutions, lowering costs and helping information departments to work more efficiently. Furthermore, Microsoft will continue to evolve the Windows family of operating systems (Windows, Windows™ for Workgroups, Windows NT and Windows NT Advanced Server) to incorporate advanced object capabilities through OLE. These capabilities will include distributed object support; object-based system services; an easy to use object-based graphical interface; integrated OLE Custom Controls; and a new object file system.

Finally, Microsoft will continue to publish object specifications and garner feedback through the Open Process, ensuring the broad interoperability and availability of OLE on other platforms. Our goal is simple: deliver innovative, best-of-breed software based on the needs of our customers. By playing a leadership role in advancing object technology, we move closer to realizing the vision of *Information at Your Fingertips*.

## For more information

For more information on OLE 2.0, Microsoft development tools, or any Microsoft application, call the Microsoft Developer Services Team toll-free at (800) 227-4679. In Canada, call (800) 563-9048. Outside the 50 United States and Canada, contact your local Microsoft subsidiary. You can also contact Microsoft by fax at (206) 936-7329. Specify Developer Services Team, RWF on your cover sheet. A TDD/TT (text telephone) is available for the hearing impaired: call (206) 635-4948.

## The Microsoft Developer Network

The Microsoft Developer Network is an annual membership program created to support all developers who write software for the Microsoft Windows family of operating systems or who use Microsoft products for development. Members receive technical and resource information on a regular basis through three channels: the Microsoft Developer Network CD, the *Microsoft Developer Network News* and the MSDNLIB Forum on CompuServe®. For more information call (800) 759-5474, seven days a week, 24 hours a day.

# Appendix A: Technical Overview

## *A Discussion of Object-oriented Programming Languages*

Object-oriented programming (OOP) offers a different approach to custom application development and can offer benefits to information departments delivering strategic information systems. Object-oriented programming serves a different need than object-enabling system software. For instance, building OLE 2.0 applications and component objects does not require the use of object-oriented programming languages, although they can be used, if desired. When developing custom applications, object-oriented programming can help with the following:

- **Reduce programming time**
- **Increase programmer productivity**
- **Make systems easier to maintain and adapt to changing business needs**
- **Increase the quality of new applications**

Because the OOP approach is quite different than standard, procedural-based system development, information departments need proper training on object-oriented design and programming concepts before these benefits can be realized. In addition, when implementing object-oriented systems using object-oriented programming languages, taking the time to plan the system blueprint is critical. Unforeseen changes to class structures (as discussed below) can wipe out any benefits received. The decision to use pure object-oriented programming languages should be weighed carefully, on a case-by-case basis.

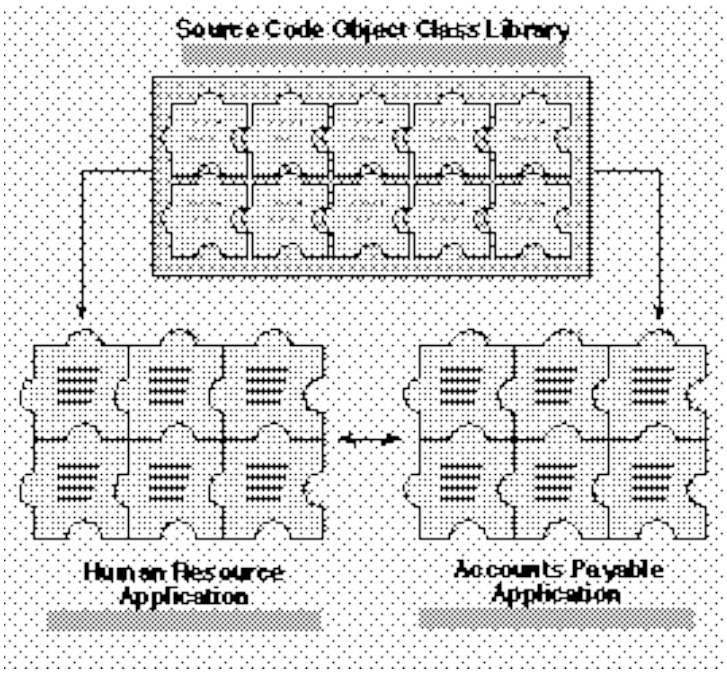
### Example: an object-oriented human-resource application

Suppose a company wants to take advantage of object-oriented programming to create a human resource application to keep track of employee records. The designers might start by defining an "employee object" that would contain specific data such as name, address, tax exemptions and pay level; then they also would incorporate specific employee methods, such as issuing a paycheck; updating benefits information; and processing tax forms. All of the sub components in this application would also be represented as objects. For example, the paycheck would be represented as a paycheck object, and the process of printing the check and making the appropriate accounting and audit-trail entries would be methods of the check object. Once designed, the objects themselves would be implemented as source code object *classes*. These classes are object definitions which are dependent on the particular object-oriented language being used.

Because of the modularity objects achieve, they can often be more easily reused in other applications than standard procedures and functions. For example, if a check object is written for the human resource application, it can be reused for an accounts payable application as well.

If designed properly, classes of objects can be reused across many different applications, and corporations can develop class libraries for this purpose. Once defined, these classes can become source-code building blocks for other applications, reducing the amount of time and money needed to deliver new systems. Initial object-oriented systems involve more programming, since class libraries are relatively sparse at first. Over time, less time is spent programming new objects. Instead, programmers can browse existing class libraries and select objects that meet their needs.

***Reusing Source-Code Objects Across Multiple Applications***




There are drawbacks to this approach. Class libraries are highly dependent on their inherent programming language and implementation, which limits their flexibility for reuse in other applications. As described in Appendix B, the packaging of objects as binary, component objects based on COM (with OLE 2.0-enabled services), can eliminate many of the practical limitations of reusing source-code class libraries. For example, a library of OLE 2.0 Custom Controls provides "packaged" binary objects that can be reused in any OLE -enabled development tool or container application.

*As objects are defined for one application, they can be incorporated into class libraries for future reuse in other applications. These objects are shared as source code, so they are implementation and language dependent, however.*

## Application frameworks

It might be expected that as class libraries are developed for reuse, there could even be a market for them. In fact, *application frameworks* are just that - commercially available class libraries that can make it easier to write new programs. Not only do application frameworks provide class libraries that reduce the amount of programming, often they make programming easier by *abstracting* interfaces to complex computing resources. For example, the Microsoft Foundation Classes (MFC) is a complete library of classes that can be used to build Windows applications in C++. These classes have high-level interfaces that are easier to learn and implement than low-level application programming interfaces (APIs). The

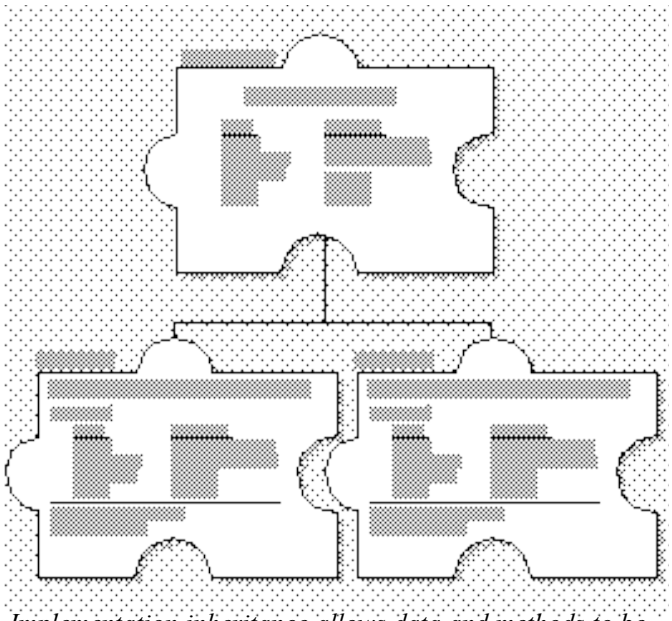
MFC classes also provide data processing and common routines that Windows applications require, substantially reducing the amount of required programming.

Application frameworks such as MFC can also abstract interfaces to other computing services such as print services; database services, and messaging services. For example, MFC provides much of the code needed to write applications using Open Database Connectivity (ODBC), a standard technology for accessing heterogeneous databases. MFC also includes full support for OLE 2.0, allowing corporations to more easily create their own OLE component business objects that can be fully integrated with other OLE-enabled packaged software.

## Implementation inheritance and class hierarchies

*Implementation inheritance* allows data and procedures of an object class to be defined once, and reused by subclasses of objects that enhance or modify the functionality to meet more specific needs. For instance, in the previous human resource example, the application might need to have the capability to differentiate permanent employees from temporary employees. While much of the data and processing would be common between these two types of employees, some processing might be unique, such as tax reporting. Through implementation inheritance, all of the common data and processing can be defined as a superclass called "employee". Subclasses of objects can then be defined for "temporary employee" and "permanent employee", with each automatically inheriting all of the data and processing functionality of the superclass "employee". Specialized methods can then be defined for tax reporting for temporary and permanent employees in order to handle these unique needs. This leads to structures called *class hierarchies*.

***Implementation Inheritance and Class Hierarchies***



*Implementation inheritance allows data and methods to be defined once, and reused by subclasses of objects. This can reduce required programming. Pictured above, specialized tax processing methods have been implemented for permanent and temporary employees. These objects, however, have also inherited much of their functionality from a superclass.*

Through a technique called *overloading*, the names for these two specialized tax processing methods can be the same, even though each method works differently depending on the object being processed (the *receiving object*). For example, the methods for processing a tax report for permanent and temporary employees might each be called "Process Taxes". The objects themselves know to carry out the call (known as a *message*) to "Process Taxes" in their unique way. Overloading can significantly reduce the number of function names that programmers need to invent and learn, and focuses application development around a language that closely follows the way we think. This hiding of alternative processing behind a common interface is called *polymorphism*- which literally means "many forms." Through polymorphism, entire families of objects can share the same method names, greatly simplifying application development. This makes applications easier to maintain, and allows other applications to reuse large portions of already developed code by tailoring objects to meet specific needs. OLE-enabled component objects, like most object-oriented programming languages, support overloading and polymorphism.

Object-enabling system software must provide for the free interaction of objects written by different companies in different programming languages. Hence, the use of uncontrolled implementation inheritance, which creates implicit dependencies between different objects through class hierarchies, should not be allowed in object-enabling system software. The COM specification for OLE-enabled component objects does not allow these implicit relationships between objects. These dependencies would limit the robustness of the objects and the entire system. For example, updating a superclass (often called a base class) object without updating a subclass object could break the system. The implementation dependencies would also greatly limit the freedom to choose objects written by different companies in different programming languages. If COM allowed uncontrolled implementation inheritance, there would be no guarantee that objects supplied by different vendors or different programming teams would work together. Furthermore, updates to objects would have to be coordinated between different companies, which can be highly problematic.

OLE-enabled component objects can, however, be easily reused across different applications through *component inheritance*- which is simply a way for an object to easily call on another object to provide a service. In this way, programmers need not re-implement code that has already been written. Instead they simply call on another component object to provide the functionality. At the same time, component inheritance is completely controlled and does not create any of the implicit dependencies between objects that result from the use of uncontrolled implementation inheritance.

Similarly, in programming custom business objects, it is important to understand that the use of implementation inheritance to achieve code reuse, while useful in some circumstances, also has some limitations. For example, if a fundamental change is required to a base class, any applications which have defined subclasses to this base class may stop working or require significant changes themselves. Class hierarchies are most effective when controlled centrally by a small group of programmers, and are not as useful when implemented in distributed systems; systems integrating objects written in different programming languages; or systems made up of diverse objects supplied by different vendors. As discussed in Appendix B, objects programmed as OLE-enabled component objects can be easily reused in all of these situations without these drawbacks.

While OLE 2.0 and the underlying Component Object Model are robust specifications meeting the requirements for object-enabling system software, some proposed system object models attempt to straddle the border between a true system object model, and object-oriented programming technology. For example, IBM SOM/DSOM and the proposed Apple OpenDoc specification (which will draw on IBM SOM for its object model), both allow uncontrolled implementation inheritance as a way to allow programmers to reuse the implementations of other objects that have already been defined. While this may be convenient from a programming perspective, the end-result will be inter-dependencies between objects which will prevent objects from being freely exchanged between different vendors' implementations, and autonomously upgraded in distributed object systems. There are many other aspects of object-enabling system software technology that are not addressed by the IBM SOM/DSOM technology or the proposed Apple OpenDoc specification. For a closer look, please see the accompanying insert comparing these technologies.

# Appendix B: Technical Overview

## *Object-oriented Programming: How OLE 2.0 and the Component Object Model Relate*

During the past few years, increasing attention has been paid to object-oriented application design and programming languages. To some, these tools have been the very essence of object technology. OLE and the underlying Component Object Model go beyond the realm of programming languages and specific object-oriented development techniques. These tools and techniques can, however, play an important role in OLE-enabled application development. In fact, object-oriented languages and programming techniques can be seamlessly integrated with the development of OLE-enabled applications, and can make the development effort faster and easier. For example, the Microsoft Foundation Classes help programmers build OLE-enabled applications by providing source-code object classes with packaged OLE functionality. Developing custom OLE-enabled component objects and/or integrating packaged OLE-enabled applications through Automation, however, does not *require* the use of object-oriented programming languages, although they can be used if desired. As object-enabling system software, OLE and COM are language independent.

### Using OLE Automation to integrate component software

It is important to understand the distinction between using OLE Automation to integrate applications into custom solutions, and actually programming your own OLE objects. To use OLE Automation simply requires any programming tool which supports OLE Automation programming (called *Automation Controllers*), and requires no technical knowledge of object-oriented programming, the OLE programming interface, or the Component Object Model architecture. Instead, the developer only needs to learn how to use the object-based interfaces (commands) that allow Automation Controllers to access the capabilities of packaged applications. Documentation on these interfaces is provided by the application vendor, in much the same way vendors provide documentation for any application-specific macro language. With this documentation, developers can then use their favorite programming tool (Automation Controllers) to drive packaged OLE-enabled applications by making calls to the object-based commands.

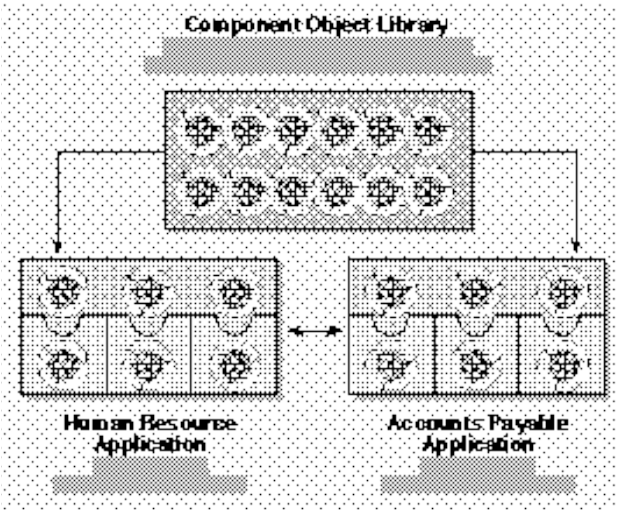
OLE Automation, in short, creates applications that "listen," and programming tools are used to "talk" to these applications, directing them to accomplish a processing task. For example, to automate the creation of a chart in a spreadsheet application from a Visual Basic program, the developer need only learn how to use the chart object's object-based interfaces (commands). Then a custom program can be developed using Visual Basic to create and manipulate a spreadsheet chart in any manner supported by the spreadsheet application. Many popular high-level development tools and programming languages already have or will soon have built-in support for using OLE Automation in this manner.

In fact, if an application's macro language supports "talking" to other applications through OLE Automation (i.e. it is an Automation Controller), then the macro language itself can be used to program calls to other applications. For instance, a developer or system integrator can use a word processor's macro language to directly make calls to the spreadsheet's object-based interfaces, if the word processor's macro language is an Automation Controller. Thus, full programming languages such as C or Visual Basic are not necessarily required to integrate OLE-enabled applications through Automation. Applications can be programmed to use each other's services directly through built-in macro languages. A good example is Visual Basic for Applications, which is available today as a macro language for Microsoft Excel 5.0, and will become the common macro language for all Microsoft Office applications.

## Developing your own OLE 2.0 applications and component objects

The availability of a wealth of OLE-enabled component software (including finer-grain component objects such as OLE Custom Controls, and larger-grain applications supporting OLE Automation), combined with the ability to easily integrate this software with custom applications, will allow corporate developers to deliver highly tailored, fully functional business solutions with a greatly reduced amount of programming. Of course corporations and systems integrators will still need to develop some specialized business processing. Today, corporate developers and systems integrators have the choice of programming custom processing logic using a traditional language such as COBOL, or an object-oriented programming language such as C++ or SmallTalk®. The advantage of using object-oriented languages over traditional languages is that in some cases object-oriented source code can be more easily reused in future applications requiring similar functionality (for instance, through class libraries and implementation inheritance).

***Reusing Software Through Component Object Libraries***




*With OLE, organizations can build up libraries of component objects, as opposed to just class libraries, and freely use these objects in multiple applications, using many different development tools and programming languages.*

OLE and the Component Object Model now give corporate developers and system integrators another choice that has distinct advantages in many situations. That choice is to program the business logic (using any programming language, object-oriented or not) as a component object supporting one or many OLE features such as Visual Editing, drag-and-drop, or Automation. In many cases this choice will empower IS organizations to deliver greater value. Some of these cases are listed below:

**1) When the developer wants the object to be able to execute in a distributed fashion across a network, without having to program any networking code.** An object-oriented programming language alone cannot give an object this ability. But all OLE-enabled component objects will have this capability when OLE with distributed capabilities becomes available, no matter what language was used to program them.

**2) When the developer wants the object to have all the end-user capabilities of an OLE object - such as incorporation into other applications and compound documents, Visual Editing, and drag-and-drop.** For instance, if a corporation is developing a custom order-entry system, by programming an invoice form with OLE server capabilities, users will be able to drag this form into word-processing documents, spreadsheets, mail systems (i.e. an email outbox), or any OLE-enabled container application. If the invoice form was also programmed as an OLE container, then any other OLE object could be placed within it. These capabilities are for the first time available through the OLE system software.

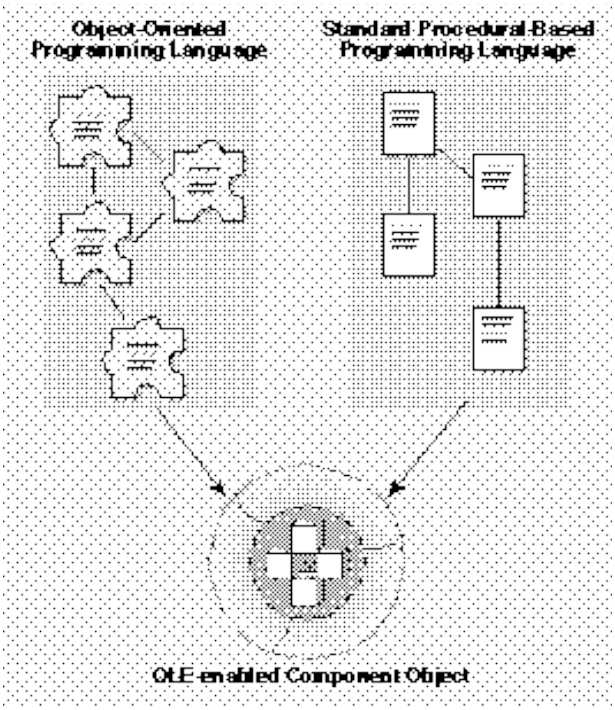
**3) When the developer wants to expose the custom object's services through OLE Automation.** Then, these custom services can be called from any programming language or any macro-language supporting OLE Automation programming (Automation Controllers). The advantages of this approach are significant:

- **Any programming language can be used to access the object's services.** The commands used to program these services are defined by the organization as a set of object-based interfaces. These interfaces can then be documented, and other development teams can reuse the object using any programming language or development tool they choose. In fact, the object's interfaces can now be called directly from any packaged application (such as spreadsheets, word processors, vertical-market applications, etc.) which has a macro language that is an Automation Controller. This not only frees corporate programming teams to use different programming tools while sharing objects across distinct development projects, it also allows corporate developers and system integrators to effectively build extensions into a packaged application. These extensions can then be freely accessed through the packaged application's macro language, and by other OLE-enabled applications.

- **The developer of an OLE-enabled component object can completely shield the object's users from the source code and implementation details of the object.** Users of OLE component objects do not need to understand, use, link to or recompile any source code whatsoever. Instead, the object is simply compiled once by the object developer, and it can then be freely reused by any other application. This eliminates the class hierarchy and source code dependencies that are prevalent in both object-oriented languages and standard procedural-based languages when reusing object classes or procedural-based source code libraries. OLE-enabled component objects are more robust, since component objects get the advanced versioning control inherent in the COM architecture. Also, with an OLE-enabled component object, it is possible to enhance and modify an individual object's functionality without recompiling and redistributing all the applications that use the object. Again, none of the object's source code has been incorporated into the applications using it, and the object itself is not dependent on any implicit class hierarchy relationships, so changes to its implementation can't break other applications.

***Creating OLE-Enabled Component Objects***

## Using object-oriented programming tools to reduce development time




The Component Object Model is completely language independent, and corporate developers can program custom business objects as OLE-enabled component objects using any programming language they want. Today, any compiler or language supporting calls to Windows-style functions can be used to create OLE 2.0-enabled component objects. In addition, application frameworks such as the Microsoft Foundation Class Library can hide much of the detail and significantly reduce the required C/C++ programming for building custom component objects.

If a corporation makes the choice to move to an object-oriented development environment, these environments can also be used to program custom OLE 2.0 objects as discussed above, in combination with commercially available frameworks. In many cases, however, and increasingly over time, the functionality required from the business object will be commercially available as a component object (such as an OLE Custom Control). It can then be purchased in packaged form for a price much less than it would cost to develop. This is one of the primary goals of component software and the Component Object Model architecture. Through OLE Automation, the purchased object can be easily integrated into a custom application using any number of high-level programming tools and development environments that support Automation Controller capabilities.

*OLE-enabled component objects are language independent, and can be created using traditional languages, as well as object-oriented programming languages and application frameworks such as C++ and MFC.*

The languages and development tools used to accomplish this integration are completely up to the particular organization. If the development team is comfortable with object oriented programming, and feels it offers significant advantages, it is free to use an object-oriented programming tool to build the integrating code. However, the team may also want to use a more traditional language such as COBOL.

Whether integrating existing OLE-enabled component objects and applications through Automation, using OLE Custom Controls, or building custom OLE-enabled component objects from scratch for future reuse, OLE and the Component Object Model create choice. In all of these cases, development time is reduced, and IS departments are empowered to deliver more flexible, higher quality applications using the development tools they choose.

## For more technical information

An excellent source of information on the Component Object Model and OLE 2.0 is *Inside OLE 2.0*, by Kraig Brockschmidt. This book is available at most major bookstores, and directly through Microsoft by calling toll-free (800) MSPRESS. *Inside OLE 2.0* presents a complete guide to programming component objects using C++. To obtain the OLE 2.0 Software Developer's Kit (SDK), call the Microsoft Developer Services Team toll-free at (800) 227-4679. If you require TDD/TT (text telephone) for the hearing impaired, call (206) 635-4948. In Canada, call (800) 563-9048. Outside the 50 United States and Canada, contact your local Microsoft subsidiary. You can also contact Microsoft by fax at (206) 936-7329. Specify Developer Services Team, RWF on your cover sheet. You are also strongly encouraged to join the Microsoft Developer Network.

# Appendix C: Glossary of Terms

Object-technology has introduced many new terms and acronyms to the software industry. The following glossary should prove useful in understanding these terms and how they relate to delivering business solutions.

***Abstraction***
Abstraction is the basis for a well constructed object-oriented system. It is the process of creating a user-defined data type, and is often referred to as "information hiding." In object-oriented programming, abstraction is used to define object classes that closely resemble real objects (such as invoices, products, etc.). These objects hide their inner complexities from users and programmers, making them easier to understand and use. *Application frameworks* also use abstraction as a way to make programming easier. Instead of learning complex, low-level application programming interfaces (APIs), programmers can use the easier-to-understand, abstracted interfaces provided by the framework.

***Automation***
Automation is one of the features of *Object Linking and Embedding 2.0*. Automation is the ability for one application to dynamically use the services of another application, without having been specifically designed to do so. For instance, a word processing application, through OLE Automation, can be customized by a user, corporate developer or system integrator, to interface with any OLE-enabled spreadsheet application to recalculate cells, draw a graph, or perform any service that the spreadsheet supports. This ability can "automate" processing tasks for users. Applications export their internal services through abstractions called *Automation Objects*. Automation is a breakthrough in object technology, since it allows independent, packaged applications to be easily integrated into custom business solutions.

***Automation Controller***
An Automation Controller is a development tool or application that can drive OLE applications through Automation. Today, Microsoft Visual Basic, Visual Basic for Applications, and Microsoft Visual C++ are all Automation Controllers. Numerous third party development tools available today are also Automation Controllers. In the near future, a corporation will be able to pick from even more development tools and programming languages to get Automation Controller capabilities.

***Automation Objects***
An Automation object is component software which supports OLE Automation. For instance, Microsoft Excel charts are Automation objects because they can be created and manipulated ("automated") by other applications through OLE Automation. OLE Custom Controls are another form of automation object. See also, *Object Linking and Embedding 2.0, Automation, and OLE Custom Controls.*

***Aggregation***
Aggregation is an object-oriented technique that allows individual objects to be grouped together to form a meta-object which provides all of the interfaces (methods) of its constituent objects. It can be used as an alternative to classical *inheritance (implementation inheritance)* in certain cases, since aggregation does not create an implicit dependency between the different objects which make up the meta-object. The Component Object Model supports aggregation.

***Application Framework***
Application frameworks are sets of objects that provide packaged functionality and programming interfaces to accomplish specific tasks. For instance, the Microsoft Foundation Classes include a set of C++ classes optimized for writing Windows-based applications, as well as classes that make it easy to use many different computing resources. Frameworks are an important aspect of object technology, since they can reduce development effort by providing interfaces that are easier to learn and use than low-level APIs. They also reduce the amount of code that needs to be written since the classes provide packaged functionality.

### *Class*
A class is an abstract data type defining the data and methods for a specific type of object. Programmers use classes to define instances of objects within their programs. As an analogy, "cat" is a class of mammals, while your neighbor's cat Waldo is a specific instance (object) of the class cat. In object-oriented programming languages, the object is implemented as a source-code template. This template is used to create specific instances of objects within programs.

### *Class Hierarchy*
A class hierarchy is a group of superclasses and subclasses that are related through an inheritance tree. For instance, the class "cat" has logical subclasses including lions, tigers, and house cats. Each of these subclasses inherits certain common characteristics (such as agility), but each also has specific features and abilities of its own. In object-oriented programming, class hierarchies are used to identify common data and procedures once as a superclass, and then enable the superclass to act as a template for related object types. Subclasses of the superclass can then be defined and tailored for specific needs without having to write completely new definitions from scratch.

### *Common Object Model*
See also the C*omponent Object Model*. The Common Object Model is an open architecture for cross-platform development of client/server applications based on object-oriented technology. Recognizing the need to allow objects on different types of operating systems to interact, Microsoft and Digital Equipment Corporation are developing the architecture to allow interoperation of OLE 2.0 and Digital's multi-platform object system, ObjectBroker. The Common Object Model is the functional equivalent of the Component Object Model for UNIX-based platforms that today include SunOS, IBM AIX, HP-UX, ULTRIX, OSF/1 and OpenVMS. The Common Object Model defines a common DCE RPC-based protocol and a subset of core OLE 2.0 functions that will be supported by Digital and other interested companies within their products. The Common Object Model is a direct outgrowth of the *Component Object Model* and provides full upward compatibility with OLE 2.0. In addition to the services already defined in OLE 2.0, ISVs, corporate developers and system integrators will be able to create new types of objects and interfaces that are both source-code portable across platforms, and interoperable across a network.

### *Common Object Request Broker Architecture (CORBA)*
CORBA is a specification produced by the Object Management Group, which attempts to define some common ground for different object models to interact in a distributed fashion. Because it is a functionally limited specification, however, it cannot guarantee interoperability between products. Today, no two CORBA-compliant products provide basic object interoperability through the CORBA architecture. There are, however, numerous products that claim to be "CORBA-compliant." Any product that claims CORBA-compliance should be evaluated carefully, since this compliance is ill-defined and cannot provide a standard for interoperability.

### *Component Inheritance*
Component inheritance allows OLE component objects to be easily reused in different applications, without creating implicit relationships between objects. This avoids the problems of uncontrolled *implementation inheritance*, which creates dependencies between objects. OLE objects, becuase they suppport component inheritance (while preventing uncontrolled implementation inheritance), can be freely exchanged between different vendor implementations, and/or upgraded without breaking existing applications. Component inheritance is simply the use of an existing *component object* to supply functionality for a new object. This reduces programming effort, since it eliminates the need to re-implement existing code. Programmers can use existing component objects to supply functionality. For instance, a programmer can easily reuse OLE Custom Controls in many different applications, saving significant development time. All component objects support reuse through component inheritance. See also, *implementation inheritance* and the *Component Object Model.*

### *Component Object Model*
The Component Object Model, or COM, is a standard mechanism for objects written by different companies in different programming languages to interact. The COM infrastructure will also support remote object connections (across a network) in a totally transparent fashion. COM defines a binary object interface, and allows stand-alone, packaged applications to dynamically explore and use services provided by other COM-based applications. COM is the basic "wiring and plumbing" for all OLE 2.0 features. For instance, COM allows component software applications to be integrated into larger business systems through *OLE Automation*. The Component Object Model provides the most advanced object features available in any system object model. These features include robust object versioning control, a complete logical thread model to prevent object deadlocks, *globally unique identifiers* to prevent object name collisions, the ability to seamlessly share objects across different address spaces, seamless support for distributed objects, language independent binding, support for object security, and many more. OLE 2.0

provides extensive object services built on top of COM, including complete support for compound documents, and cross-application object scripting (Automation). In addition to the services already defined in OLE 2.0, ISVs, corporate developers and system integrators will be able to create new types of objects and interfaces that are both source-code portable across platforms, and interoperable across a network. Today there are over 1.5 million component objects being used in desktop business applications. See also, *Object Linking and Embedding 2.0* and *Object-Enabling System Software.*

### *Component Object*

Component objects are objects that support the *Component Object Model* specification. Component objects can dynamically explore the capabilities of other component objects, and use their services through OLE Automation. This is a breakthrough for the software industry, since component objects allow independent, packaged objects to be seamlessly integrated by users and system integrators, no matter what company designed the objects or which programming language was used to program them.

### *Component Software*

Component software is an application that contains one or more *component objects,* which can freely interact with other component software through OLE capabilities. Examples include OLE -enabled applications such as the Microsoft Office applications, and soon packaged, commercially available OLE *Custom Controls*. The component software market will offer a more productive approach to assembling (versus building from scratch) complete business solutions.

### *Compound Documents*

Compound documents are documents which contain multiple data types. Often, the different types of data have been created by different applications, and embedded into the document. For instance, a report document is a compound document if it contains an embedded spreadsheet table. OLE enables the creation of rich, compound documents by treating application data as objects which can be freely moved and manipulated between applications.

### *Container*

A container application is an OLE-enabled application that can store embedded or linked objects provided by OLE *server* applications. When dragging a spreadsheet chart into a word processing document, for example, the spreadsheet application is the OLE server, while the word processor is the OLE container. OLE 2.0-enabled applications can be both object containers and object servers.

### *Distributed Object System*

A distributed object system is a system in which objects located on different machines cooperate to accomplish a common task. Any distributed system with different processes executing and communicating across a network could probably be called a distributed object system, since the definition of object is not precise. In this sense, distributed object systems are really client-server systems. However, in terms of OLE-enabled component objects, the term *distributed object system* can be more narrowly defined as a set of two or more component objects that are running on different physical computers, but cooperating through the OLE interfaces *as if they were on a single machine*. For instance, an Automation Controller could just as easily "drive" an OLE-enabled application running on another machine as it could an OLE-enabled application running on the same machine. The Automation Controller and the OLE-enabled application being controlled will actually be running on different CPUs, with network transparency provided by the Component Object Model. This makes it possible to split applications or processes into logically distinct tasks that can be executed remotely and concurrently on separate machines. Any OLE 2.0-enabled application can be run in "client-server" configuration, even though it has not necessarily been designed for this capability (i.e., OLE 2.0-enabled applications will get networked, client-server capabilities for free!).

### *Encapsulation*

Encapsulation is the technique of combining data and processing logic within self-contained software objects. These objects hide their inner complexities from programmers and users. The encapsulated functionality of the object is accessed through well defined interfaces made up of *methods*.

### *Globally Unique Identifiers (GUIDs)*

Globally Unique Identifiers are IDs assigned to OLE *component objects*, and are generated through a sophisticated algorithm. The algorithm guarantees that all component objects will get unique IDs, avoiding any possibility of a naming conflict, even is systems with millions of objects (supplied by many different vendors). Software developers can easily generate GUIDs for their component objects through special software provided in the OLE 2.0 SDK.

***Inheritance***
Inheritance is the mechanism that allows specific customized objects to be defined from more general definitions. See also *component inheritance* and *implementation inheritance.*

***Implementation Inheritance***
Implementation inheritance is the mechanism that allows subclasses of objects to be defined by using general, superclass object definitions as templates. Common data and methods are defined at the superclass level, and additional, specialized data and/or methods are defined for each subclass. Implementation inheritance is thus derived from *class hierarchies*. While useful for object-oriented programming languages, traditional implementation inheritance should be avoided in object-enabling system software, because it creates implicit inter-dependencies between objects. This prevents objects from being freely interchanged between different vendors' implementations, or autonomously upgraded in *distributed object systems*.

***Interface***
Generally speaking, an interface is simply a mechanism for different pieces of software to interact. For instance, application programming interfaces (APIs) are provided with operating systems (to access system-level services from programming languages); database management systems (to access SQL database services); and any number of other types of applications and system software. As defined by the Component Object Model and OLE, however, an interface is a collection of methods, and defines a strict "contract" between objects. Objects can only access each other through a collection of standard interfaces, and an application supporting an interface must support all of the methods defined by that interface. The interfaces are binary standards, so users are guaranteed that objects talking to each other through the COM interfaces will seamlessly interoperate. Because OLE 2.0 is based on COM, OLE 2.0 applications are guaranteed to work with each other no matter who programmed them, or in what language they were programmed.

***In-Place Activation***
See *Visual Editing*.

***Message***
In pure object-oriented programming terms, a message is the invocation of an object's method, and is the only way objects can interact. When one object wants to call on another object's functionality, it is said to "send a message" to the object. The object sending the message is sometimes called the *sender*, and the object receiving the message is sometimes called the *receiver*. The message consists of the method name, and any parameters required by the method. In the Component Object Model, collections of methods are called an *interface.* The term message is also often used to denote a system event, especially in GUI-based systems. For instance, a mouse-click in the Windows environment is said to "generate a message." This should not be confused with the term as applied to object technology.

***Method***
A method is a logical operation provided by an object. Operations performed on objects are defined as "methods of the object." To invoke a method, an object sends a *message* consisting of the receiving object and the name of the specific method to invoke. The name of the method is sometimes called a *selector.* Messages are the only way for objects to interact.

***Object***
See *Object Technology*

***Object Embedding***
See *Object Linking and Embedding 2.0.*

***Object-Enabling System Software***
See also the *Component Object Model* and *Object Linking and Embedding 2.0*. Object-enabling system software is software that occupies a layer between applications and the operating system, and defines a common object model for all applications and other system software to use. Such a common object model can break down the barriers between different objects and packaged applications, allowing them to freely interact. The operating system itself can also take advantage of these common object capabilities, allowing better integration between application objects and system services, including the user-interface, the file system or any computing resource. OLE is the most advanced and widely used object-enabling system software available, and allows diverse objects to interact on a single machine, and soon across a network. Today, OLE is available for the Microsoft Windows family of operating systems, and Microsoft is also providing OLE 2.0 for the Apple Macintosh System 7 operating system.

Furthermore, OLE-enabled component objects will also be available on many other operating system platforms, major versions of UNIX, VMS, and other platforms. OLE is based on an underlying object model, the *Component Object Model*, which defines a binary interface standard for objects. COM provides the basic "wiring and plumbing" for all OLE 2.0 applications, and has been designed for distributed capabilities. In addition to the services already defined in OLE 2.0, ISVs, corporate developers and system integrators will be able to create new types of objects and interfaces that are both source-code portable across platforms, and interoperable across a network.

### *Object Linking*

See *Object Linking and Embedding 2.0.*

### *Object Linking and Embedding 2.0 (OLE 2.0)*

OLE 2.0 is a set of system services that provides a powerful means for applications to interact and interoperate. Based on the underlying *Component Object Model*, OLE 2.0 is a breakthrough in object-enabling system software. Through OLE *Automation*, an application can dynamically identify and use the services of other applications, making it possible for corporate developers and system integrators to build powerful business solutions using packaged software. OLE 2.0 also makes it easy to create documents consisting of multiple sources of information from different applications.

Applications that accept objects from other applications are called *containers*, while the application providing the object is called a *server*. Through OLE *object linking*, objects created in one application can be linked into container applications. As the linked object is changed or revised by the server application, it is automatically updated in any container applications. Through OLE *object embedding*, a distinct copy of the object is made, and then embedded in the container application. Changes to the original data are not reflected in the embedded object. Through OLE *Visual Editing*, embedded and linked objects can be directly edited within the container application without switching to the server applications. Through OLE *drag-and-drop*, information can be selected in a server application and dragged into a window running a container application. Today, OLE 2.0 is available for the Windows Family of operating systems (32-bit OLE 2.0 will be available with the upcoming upgrade for Windows NT 3.1). Microsoft will soon release OLE 2.0 for the Apple Macintosh, and through an agreement with Digital Equipment, OLE will be available on many other platforms, including most versions of UNIX.

### *OLE Custom Controls*

OLE Custom Controls are a special form of component *Automation Objects*. Custom Controls are similar to Visual Basic custom controls (VBX's), except that their architecture is based on OLE 2.0. This means that, unlike VBX's, they can be freely plugged into any OLE-enabled development tool, application, and eventually the Windows operating system itself. These controls will be provided by numerous software vendors, and will offer packaged functionality ranging from database access to mainframe connectivity to workgroup messaging functionality. Both 16-bit and 32-bit components can be developed using the same OLE Custom Control source code (simply recompiling) under Windows 3.1, the Windows NT operating system and future versions of Windows. Because it is based on OLE, a cross-platform technology, the new OLE Custom Control architecture also opens up the possibility of OLE Custom Controls being available in the future on additional operating systems such as the Macintosh and UNIX.

### *Object Technology*

A broad term which implies the use of "objects" to 1) analyze; 2) model or design; and/or 3) implement some aspect of a computer system. In terms of actual application implementation (as opposed to object-oriented analysis or design), objects are self-contained software modules that encapsulate both data and processing logic, and can only be accessed through well defined interfaces. While the basic goals of object technology are to increase system modularity and hence flexibility; reduce programming time; reduce maintenance costs; and increase ease of use, it is useful to distinguish between object-oriented programming technology and object-enabling system software technology. See also *object-oriented analysis*, *object-oriented programming*, and *object-enabling system software*.

### *Object-Oriented Programming (also known as OOP)*

Object-oriented programming, as opposed to procedural programming, involves the use of both object-oriented design, and an object-oriented programming language such as C++ or SmallTalk. Instead of consisting of sets of data loosely coupled to many different procedures, object-oriented programs consist of software modules called objects, which encapsulate both data and processing while hiding their inner complexities from programmers (and hence other objects). This can make object-oriented programs more flexible and easier to maintain. Through *implementation inheritance* and *polymorphism*, objects can be reused across multiple systems, reducing programming time. OOP does not address the need for application interoperability, or object interaction across application boundaries. Instead, *object-enabling system software*, such as OLE, is required to achieve these critical

benefits. Also, object-oriented programming need not be applied to the implementation of systems/applications designed through object-oriented analysis and design techniques.

***Object-Oriented System Analysis and Design (also known as OOA for Object-Oriented Analysis)***
The process of modeling a system by breaking it down into a series of objects that closely resemble real, physical objects. This design model is becoming particularly relevant as organizations re-engineer their business processes, and deploy new systems to alter and improve workflow. Through object-oriented analysis and design, for instance, a claims processing system might be modeled as a series of business processes, represented as objects such as the claims form itself, and the different departments, people, data and tasks that need to be included in the successful processing of a claim. Based on breaking the entire process down into individual components, organizations can more readily detect weaknesses, as well as ways to improve and/or automate the system. Note that the use of an object-oriented programming language is not necessary to achieve these benefits. The OLE object-enabling system software will, however, allow organizations to deliver better distributed workflow systems because such software enables the smooth integration of many different types of applications (including applications communicating across a network). Such integration will help organizations to build more capable, flexible systems, and deliver these systems at a lower cost.

***ObjectBroker***
A product sold by Digital Equipment Corporation that provides a mechanism for building distributed, client-server systems incorporating objects running across multiple platforms. The ObjectBroker facilitates the network communication and interoperability of the different objects. Digital's ObjectBroker will incorporate the *Component Object Model*, allowing OLE-enabled applications to access and use objects running on a variety of UNIX platforms and VMS. OLE support for many other platforms is also planned.

***Object Registry***
A database listing the OLE-enabled component objects available on a system. It is updated whenever an object is installed or executed. Container applications know what objects are available by looking at the object registry.

***OpenDoc***
OpenDoc is a specification for a compound document architecture that is being formed by the joining of several different technologies supplied by Apple (the base OpenDoc architecture, the Bento file system and the Open Scripting Architecture) and IBM (the System Object Model). The development effort for combining these technologies has been split between key consortia members, including Apple, IBM, Borland, and WordPerfect. Today, OpenDoc is still a paper specification, not available on any platform. See also, the *System Object Model/Distributed System Object Model.*

***Open Process***
The formal review process which Microsoft uses to ensure the quality, interoperability and acceptance of new system software innovations. The Open Process allows industry experts, corporations, OEMs, and independent software vendors to participate in the development and direction of the Windows platform, through specification and design reviews. OLE 2.0 was refined through an Open Process through which major software vendors (including Apple Computer, Borland International, Lotus Development Corporation and the WordPerfect Corporation) participated in open design reviews. Additionally, preliminary OLE 2.0 specifications were distributed to more than 150 different software vendors for open review and feedback.

***Overloading***
A technique used in object-oriented systems (both in programming languages and in object-enabling system software) whereby different object classes use the same method name even though the method definitions may differ. This reduces the number of method names that need to be created and memorized by programmers, and allows sub-classes of objects to by tailored to meet specific needs. Overloading can be fully used (and is) in OLE-enabled component objects to achieve *polymorphism*.

***Polymorphism***
Polymorphism is a feature of object-oriented software technology (both programming languages and object-enabling system software) that allows different objects to be accessed through the same interface, although each object can perform its own custom processing when invoked through this interface. Thus, alternate processing is "hidden" behind a common interface. For instance, *overloading* leads to polymorphism.

***Properties***

The attributes associated with a *component object*. For example, a chart has the properties of color and type, to name a few. In the next version of Windows, documents, files and other objects will be endowed with specific identifying properties which can be viewed through *property sheets*.

***Property Sheets***
Property sheets will be introduced with the next release of Microsoft Windows, and are forms that allow users to view the *properties* (attributes) of objects on their desktop. Property sheets will make it very easy to access critical information about an object (such as a file or printer), without opening the object or having to use a special utility to view the information.

***Receiver***
The object that is receiving a message to act on. The receiver object invokes the method implied by the message.

***Remote Procedure Call***
A mechanism through which applications can invoke procedures (and object methods) remotely across a network. Using RPC, an application on one machine can call a routine (or invoke a method) belonging to an application running on another machine. RPC is popular because it uses a familiar approach of invoking programmed procedures, making the networking details transparent to programmers. Microsoft is using a DCE-compatible RPC as the infrastructure in future versions of Windows to enable communication between OLE 2.0 applications running on different machines. Since this RPC is DCE-compatible, it will facilitate object interactions across multiple platforms.

***Sender***
An object which requests the invocation of another object's method.

***Server***
While the term server is widely used to denote a computer that provides services (such as file access or database access) to client applications, in OLE 2.0 terminology a server is an OLE 2.0-enabled application that can provide an OLE *container* application with objects. When dragging a spreadsheet chart into a word processing document, for example, the spreadsheet application is the OLE server, while the word processor is the OLE container. OLE 2.0-enabled applications can be both object containers and object servers.

***System Object Model/Distributed System Object Model***
The System Object Model (SOM) and the Distributed System Object Model (DSOM) are object models defined by IBM that are available for IBM OS/2® and IBM AIX. These object models offer some benefits, but they do not address many of the important needs that object-enabling system software should address. SOM itself, for instance, does not allow objects to communicate and be shared between different programs (even between processes running on the same machine). To achieve these capabilities, IBM provides a different approach-- DSOM. Unlike OLE and the Component Object Model, SOM/DSOM do not achieve local and distributed object support with a single, integrated object architecture. More importantly, SOM/DSOM straddle the border between object-oriented programming technology and a true system object model. SOM and DSOM allow uncontrolled *implementation inheritance:* objects can inherit source-code implementations from each other, through *class hierarchies*. While this can make programming SOM/DSOM objects somewhat faster, it also introduces interdependencies between objects. These interdependencies can prevent objects from being freely interchanged between vendor-implementations, or autonomously upgraded in distributed systems. In addition, the lack of compound document support, the lack of robust object versioning, the lack of *globally unique object identifiers*, and the lack of a logical thread model to prevent object-deadlocks are some of the other limitations of the SOM and DSOM object models. All of these issues are successfully addressed, however, by *OLE 2.0* and the underlying *Component Object Model*.

***Visual Basic Custom Control***
Visual Basic custom controls (VBX's) are a specific form of binary, packaged objects that can be created by different companies and integrated into Visual Basic Applications. But because they are based on a Visual Basic interface, they cannot be easily reused in other development tools and applications. With the introduction of *OLE Custom Controls,* a form of *component software*, these limitations are eliminated.

***Visual Control***
See *OLE 2.0 Custom Controls.*

***Visual Editing***

Visual Editing, sometimes called in-place activation, is a feature of OLE 2.0 which makes it easy to create and edit compound documents. By double-clicking an embedded object within a compound document, the menu and toolbar of the container application will change to the menu and the toolbar of that object's native application. This allows the object to be edited within the context of the document ("in-place") instead of switching between the different applications. See also *Object Linking and Embedding 2.0.*