**Component Object Model & Object Linking and Embedding**

COM is a framework for creating and using objects. COM, the Component Object Model delivers on the long-promised benefits of object technology: code reuse and off the shelf components. How does it do this? By providing a standard way to create and use components with a wide choice of tools, languages and applications.

The Component Object Model (COM) is a system technology that originated with Windows, but has begun to propagate to other platforms (the Macintosh, Compaq/Digital VMS, Compaq Digital Unix, Solaris, other Unix flavors, mainframes, etc.) as well. Its purposes for software developers are multi-fold. We’ll focus on COM technology and concentrates on its advantages, which have made it one of the most versatile and indispensable technologies of today.

***INTRODUCTION***

# OLE (Object Linking and Embedding)

OLE stands for object linking and embedding, is a feature that allows users to create and edit documents that contain objects created by different applications.

For example, you can embed bitmap images, sound clips, spreadsheet files, and other objects in Microsoft Word documents.

The term “object linking and embedding” comes from these two types of actions for creating compound documents:

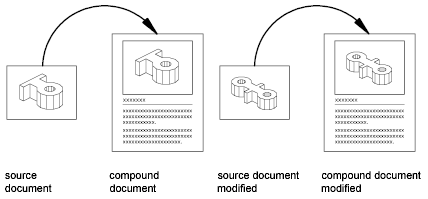
* **Linking** adds a link in a document that points to source data stored somewhere else. Linked objects are stored in the document as a path to the original linked data, usually a separate file from the container document. A linked object is a reference to information in another document. Link objects when you want to use the same information in more than one document. Then, if you change the original information, you need to update only the links in order to update the document containing the OLE objects. You can also set links to be updated automatically. When you link a drawing, you need to maintain access to the source application and the linked document. If you rename or move either of them, you may need to re-establish the link.

Figure 1. Linking

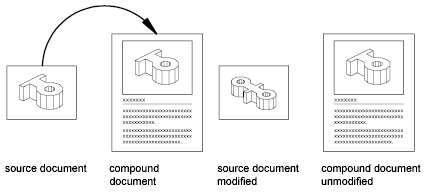
* **Embedding** on the other hand, adds one document directly to the other. Embedded objects are stored with the document that contains them. An embedded OLE object is a copy of information from another document. When you embed objects, there is no link to the source document and any changes made to the source document are not reflected in destination documents. Embed objects if you want to be able to use the application that created them for editing, but you do not want the OLE object to be updated when you edit information in the source document.

Figure 2 . Embedding

Part of the confusion related to OLE technology comes from the fact that Microsoft has used this name for different purposes. Everything started with Object Linking and Embedding (OLE, for short) which was an extension of the DDE (Dynamic Data Exchange) model.

Using the Clipboard allows you to copy some raw data, and using DDE allows you to connect parts of two documents, Object Linking and Embedding allows you to copy the data from a server application to the client application, along with some information regarding the server, or a reference to some information stored in the Windows Registry. The raw data might be copied along with the link (Object Embedding) or kept in the original file (OLE Linking).

Its primary use is for managing compound documents, but it is also used for transferring data between different applications using drag and drop and clipboard operations. The concept of "embedding" is also used for embedding multimedia in Web pages, which tend to embed video, animation (include Flash animation), and music files within the HTML code.

**OLE 1.0**, released in 1991, was the first wildly adapted specification for developing component-document applications. It was the evolution of the original dynamic data exchange (DDE) concepts which Microsoft developed for earlier versions of Windows. While DDE was limited to transferring limited amounts of data between two running applications, OLE was capable of maintaining active links between two documents or even embedding one type of document within another.

OLE 1.0 later evolved to become architecture for software components known as the component object model (COM) which further graduated to Distributed Component Object Model (DCOM).

# OLE 2.0 , Released in early 1993, it provided a much richer compound document model (i.e. the containing multiple data types like text, video, graphics etc.), as well as OLE automation, OLE drag and drop and generic services.

OLE 2, the second version of OLE, improved on OLE 1 and expanded the support for creating compound documents. OLE 2 was based on the model known as the Component Object Model (COM). Microsoft began to recognize that OLE 2 could be used to solve other software problems and that it could be applied to other areas of software development. Microsoft saw OLE 2 as an expandable architecture to create software and, as such, decided to drop the version number.

At the core of OLE 2.0 is the Component Object Model (COM), a specification that allows developers to design interfaces that enable interaction among components. In fact, OLE 2.0 is simply a set of COM interfaces designed by Microsoft.

# ActiveX , In 1996, Microsoft renamed the OLE 2.0 technology as ActiveX. This version of OLE is commonly used by Web designers to embed multimedia files in Web pages. ActiveX is a set of programming technologies created by Microsoft that enables software components created in different languages to interact with one another in a networked environment. It evolved from the OLE development standard, which in recent years has expanded far beyond the concepts of object linking and embedding that formed the original acronym.

Like OLE, ActiveX is built on the COM programming model to support the full integration of software components. It supports Distributed COM (DCOM) for the transparent integration of those same components across distributed networks, including the Internet and intranets. However, ActiveX has been optimized for size and speed to allow developers to use subsets of the complex OLE interface to create highly interactive applications.

The terminology is changing as fast as the technology, and not all groups within Microsoft can agree on how to use the terms ActiveX and OLE. Think of ActiveX as something that was created when the "old" OLE collided with the Internet. ActiveX includes not only those Windows features built on COM family and the WinInet programming interface.

# *Overview*

The Component Object Model (COM) is the foundation of much of the new Microsoft ActiveX technology, and after five years it's become an integral part of Microsoft Windows. Now we already have COM ++ and most Windows programming will involve COM, so you'd better start learning it now. But where do you begin? You could start with the Microsoft Foundation Class classes for ActiveX Controls, Automation, and OLE, but as useful as those classes are, they obscure the real COM architecture. You've got to start with fundamental theory, and that includes COM and something called an interface.

A component software architecture from Microsoft, which defines a structure for building program routines (objects) that can be called up and executed in a Windows environment.

Some parts of Windows and Microsoft developed applications are also built as COM objects. COM provides the interfaces between objects, and Distributed COM (DCOM) enables them to run remotely.

COM was designed with C++ programming environment in mind. It supports encapsulation, polymorphism, and reusability. However, COM was also designed to be operating at compatible at the binary level and therefore is different from a C++ object. Generally, the high level programming languages such as C, C++, PASCAL, and ADA are machine-dependent. As a binary object, a COM object concentrates on its interface with other objects. When not used in the environment of its creator, an interface is exposed that can be seen in the non-native foreign environment. It can be seen because it is a binary object and therefore *not machine-dependent*. This does not require the host environment or an interacting object to know anything about the COM object. It is important to note that COM is not a programming language; it is a binary standard that enables software components to interact with each other as objects. COM is not specific to any particular programming language. COM can work with any language that can support the binary layout of a COM object. It is a programming model to facilitate the programmability related to this standard.

***What is COM?***

The "problem" is that there's no standard way for Windows program modules to communicate with one another. "But," you say "what about the DLL with its exported functions, Dynamic Data Exchange (DDE), the Windows Clipboard, and the Windows API itself, not to mention legacy standards such as VBX and OLE 1? Aren't they good enough?" Well, no. You can't build an object-oriented operating system for the future out of these ad hoc, unrelated standards. With the Component Object Model, however, you can, and that's precisely what Microsoft is doing.

What's wrong with the old standards? A lot. The Windows API has too large a programming "surface area": more than 350 separate functions. VBXs don't work in the 32-bit world. DDE comes with a complicated system of applications, topics, and items. How you call a DLL is totally application-specific. COM provides a unified, expandable, objectoriented communications protocol for

Windows that already supports the following features:

* A standard, language-independent way for a Win32 client EXE to load and call a Win32 DLL.
* A general-purpose way for one EXE to control another EXE on the same computer (the DDE replacement).
* A replacement for the VBX control, called an ActiveX control.
* A powerful new way for application programs to interact with the operating system.
* Expansion to accommodate new protocols such as Microsoft's OLE DB database interface.
* The distributed COM (DCOM) that allows one EXE to communicate with another EXE residing on a different computer, even if the computers use different microprocessor-chip families, that means different platform or hardware.

So what is COM? That's an easier question to ask than to answer. COM is a powerful integrating technology that allows you to

mix all sorts of disparate software parts together at runtime. COM allows developers to write software that runs together regardless of issues such as thread-awareness and language choice. COM is a protocol that connects one software module with another and then drops out of the picture. After the connection is made, the two modules can communicate through a mechanism called an interface. Interfaces require no statically or dynamically linked entry points or hard-coded addresses other than the few general-purpose COM functions that start the communication process. An interface (more precisely, a COM interface) is a term that you'll be seeing a lot of.

# 1. COM Objects

COM objects can be small or large. They can be written in several programming languages, and they can perform any kind of processing. A program can call the object whenever it needs its services. Objects can be run remotely (DCOM) over the network in a distributed objects environment.

# 2. Automation (OLE automation)

Standard applications, such as word processors and spreadsheets, can be written to expose their internal functions as COM objects, allowing them to be "***automated***" instead of being manually selected from a menu. For example, a script could be written to extract data from a database, summarize it and draw the graphics from a spreadsheet and place the results into a text document.

# 3. Controls (OLE controls, ActiveX controls)

Applications can invoke COM objects, called "controls," that blend in and become just a part of the program. ActiveX controls can also be downloaded from the Internet to make a Web page perform any kind of processing.

# 4. Compound Documents and ActiveX Documents

Microsoft's OLE compound documents are based on COM, which lets one document be embedded within or linked to another. ActiveX Documents are extensions to OLE that allow a Web browser, for example, to view not only Web pages, but any kind of document.

# 5. Programming Interfaces

Increasingly, Microsoft is making its standard programming interfaces conform to the COM object model so that there is continuity between all interfaces.

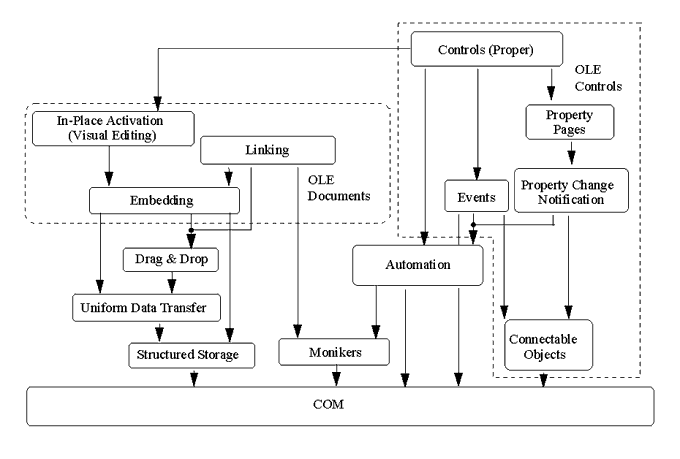


Figure 3. OLE technologies build on one another, with COM as the foundation.

# *THE ADVANTAGES AND DISADVANTAGES*

**ADVANTAGES**

* The need to build and update entire application every time is eliminated. Rebuilding a single component is enough.
* It promotes component based software development which has several advantages like ability to use prepackaged components and tools from third party vendors into an application and support for code reusability in other parts of the same application.
* COM makes it possible for different language components, which adhere to COM specifications to interact with each other.
* It helps to access components loaded in different machines on the network.
* It can segregate applications via binary firewalls or interfaces that can reduce or eliminate dependencies between primary elements.
* It provides many features to developers in industry like language independency, binary standard, wide software industry support etc.

# Object Oriented Features

All the COM objects are Object Oriented as its name implies. There are three basic Object Oriented concepts:

1. **Encapsulation**: This is implemented by the interfaces provided by the COM Objects. The interface hides the implementation details from the end user and provides the functionality to the user.

1. **Polymorphism**: This is often referred as “One Method Multiple Interfaces”. A COM object can define a single method to perform a specific operation; but that operation could be implemented through various ways.

1. **Inheritance**: When the user wants to incorporate some additional functionality to an existing COM object, he can enhance the existing COM object by inheriting a new COM object from it.

# Loose Coupling In software, which uses COM objects, one can easily replace an existing COM object with another COM object written in entirely another language as long as the signatures of the methods in both the COM objects remain same. In such a case, there will not be any change in the existing software code that uses the COM object.

# Binary Language Since most of the COM libraries are in binary language, it could be used by any application written in any language. So COM is language independent.

# Resource utilization Every COM object will be destroyed automatically as long as no client is using that object actively. This is implemented by COM using a technique called reference counting. Every COM object will maintain a reference count (the number of clients using that COM object); Once that count reaches zero (that means no clients are actively using that COM object), then that COM object will be destroyed automatically. With this approach, we can increase resource utilization in a single application. Resources such as memory will be best utilized by releasing the inactive/unused COM objects.

**DISADVANTAGES**

# Difficult COM Component Development COM components are often difficult for developers to implement. An extensive amount of code must be provided to implement a valid COM object. Furthermore, the software debugging tools for COM objects are less developed than their non-component counterparts. Since COM objects can be active on multiple computers, it is often difficult to determine the source of malfunction(s), if any.

# Divergence in Standards The COM architecture only specifies how software components communicate – not the specific interfaces they should use. Microsoft has already defined many interfaces that provide standard features such as drag-and-drop, data transfer, and persistent storage. These are not the only valid interfaces; any developer can define and implement their own custom built interface. Although this in one of the greatest strengths of COM, it is also one of its greatest weaknesses. With unchecked interface proliferation, developers will create their own closed set of interface standards that will be incompatible with the work of other developers. This problem has been partially averted through Microsoft’s use of a standard scripting interface referred to as "IDispatch." The coordinated development of interface standards would be necessary for the ongoing success of COM.

* The other major drawbacks include difficulty in integrating internet technologies and expensive, difficult, and undependable deployment.

**DCOM**

Microsoft's Distributed COM (DCOM) extends the Component Object Model (COM) to support communication among objects on different computers—on a LAN, a WAN, or even the Internet. With DCOM, an application can be distributed at locations which are required by the customer and the application. Because DCOM is a seamless evolution of COM, one can take advantage of the existing investment in COM-based applications, components, tools, and knowledge to move into the world of standards-based distributed computing. In the process, DCOM handles low-level details of network protocols so one can focus on the real business thus providing great solutions to the customers.

## COM+

It is the enhancement of the Microsoft Component Object Model (COM) that enables programmers to develop COM objects with more ease. For example, COM+ allows native C++ calls to be translated into equivalent COM calls. In addition, instead of defining COM interfaces in the traditional IDL language, they can be defined by more familiar programming syntax.

Where the Component Object Model (COM) ends, COM+ starts. COM+ extends COM to let you create components that scale better for:

* **Security in Distributed environments**: COM+ makes it easier to build components that can utilize the security subtleties of Distributed COM (DCOM) .
* **Component Services**: For event notifications, synchronization, and de-coupling of event sender and receiver (queued components, etc.).
* **Performance in Distributed environments**: COM+ makes it easier to build components that can stand the stress of high-volume instantiation requests over short periods of time.
* **Deployment in Distributed environments**: COM+ makes it easier to distribute server components or client-side components (typelibs and proxy-stub DLLs) without having to copy, unpack, register and configure each of them.

***OLE DB Architecture***

OLE DB stands for *Object Linking and Embedding Database*. Whereby linking it meant linking the object with the data source and by embedding it is meant to add the source data into the object. OLE DB is also an Application Programming Interface (API) that allows applications to access data from varied data sources. It is in fact a group of interfaces (APIs) that are used to gain access to data sources to fetch different kinds of data files like an object database, spreadsheets, personal databases, Indexed- sequential files, SQL-based DBMS files etc., which are not all necessarily SQL using file formats.

The OLE DB helps in facilitating data access for the application from the data sources. The access to the data source is possible through a set of abstractions or components in OLE DB namely; ***Data source, Session, Row sets and Command***. These four are also termed as OLE DB objects. By these four abstractions playing as a bridge between the application and data source, the client can gain access to varied (not only SQL) sources and types of data without getting into the technicalities of the process.

To fetch data from the data source, the OLEDB of an application first initializes the OLE. Followed by connection establishment with the data source. After this, a data access command is issued to be sent to the data source. Then the data sent as requested is processed and the OLE is uninitialized releasing the data source object. The OLE DB conceptually divided into two sections; Consumers and Providers. Consumers are the user applications that require the data and the providers are those who implement the APIs and in turn, fetch the required data from the data sources.

**Data Source**: SQL Server Native Client uses the term data source for the set of OLE DB interfaces used to establish a link to a data store, such as SQL Server. A data source object creates and manages the connection and contains permissions and authentications information.

**Session Objects**: A session manages a particular interaction with the data source to query and retrieve data. Each session is a single transaction. A transaction is an indivisible work unit defined by the ACID test (**Atomicity, Consistency, Isolation, Durability**). Session objects provide methods for creating commands and row sets and for creating and modifying tables and indexes. They also define transaction scope and can be used to create transaction objects, which are used to control nested transactions.

**Rowsets** and **Commands**:A rowset is a set of rows that contain columns of data. Rowsets are central objects that enable all OLE DB data providers to expose result set data in tabular form. Commands execute a text command such as a SQL statement. If the text command specifies a rowset, such as a SQL **SELECT** statement, the command creates the rowset.

Other Components Include:

* **Accessors** describes how data is stored in a consumer. It has a set of bindings (called a column map) between rowset fields (columns) and data members that you declare in the consumer.
* **Transactions** objects are used when committing or aborting nested transactions at other than the lowest level. A transaction is an indivisible work unit defined by the ACID test.
* **Enumerators** search for available data sources and other enumerators. Consumers that aren't customized for a particular data source use enumerator to search for a data source to use.
* **Errors,** any interface on any OLE DB object can generate errors.
* **Notifications** are used by groups of cooperating consumers sharing a rowset (where sharing means that the consumers are assumed to be working within the same transaction).

# ONLINE RESOURCES & REFERENCES

* MSDN LIBRARY VISUAL STUDIO 6.0
* http://www.microsoft.com/technet
* http://computing-dictionary.thefreedictionary.com (Definitions and Figures)
* http://www.howtodothings.com/computers/a1239attributesadvantages-of-com.html
* http://www.peterindia.net/COMOverview.html
* http://www.richardhaleshawgroup.com
* http://media.wiley.com/product\_data/excerpt/23/0764559

9/0764559923.pdf

* http://www.emory.edu/BUSINESS/et/com/
* http://en.wikipedia.org/wiki/Object\_linking\_and\_embeddi ng (OLE references)