OLE 2.0 Design Specification

Microsoft OLE 2.0 Design Team
15 April, 1993

Writers: Bob Atkinson, Tony Williams, Randy Kerr, Barry Potter, and the OLE 2.0 Documentation Team
Editor: Bob Atkinson
Contributors: Bob Atkinson, Kraig Brockschmidt, Douglas Hodges, Randy Kerr, Srini Koppolu, Barry Potter, Tony Williams, Craig Wittenberg, the OLE 2.0 Development Team and the OLE 2.0 Documentation Team

OLE 2.0 Design Specification

Microsoft OLE 2 Design Team
15 April, 1993

This document is the design specification for version 2.0 of Microsoft's Object Linking and Embedding (OLE) compound document architecture. It is the definitive description of the integration interfaces and infrastructure services provided by OLE, and the means by which applications exploit that functionality.

1. Introduction

Object Linking and Embedding (OLE) is a mechanism that allows applications to interoperate more effectively, thereby allowing users to work more productively. End users of OLE container applications create and manage *compound documents.* These are documents which seamlessly incorporate data, or *objects*, of different formats. Sound clips, spreadsheets, text and bitmaps are some examples of objects commonly found in compound documents. Each object is created and maintained by its server application, but through the use of OLE, the services of the different server applications are integrated. End users feel as if a single application, with all the functionality of each of the server applications, is being used. End users of OLE applications don't need to be concerned with managing and switching between the various server applications; they focus solely on the compound document and the task being performed.

The first version of OLE, OLE 1, was included as part of Microsoft Windows 3.1. It included the basic ability to create linked and embedded compound documents. OLE 2 is designed as a natural extension to the OLE 1 architecture. In developing this architecture we have made every effort to preserve and elaborate the user conceptual model introduced with OLE 1. Both users and implementors of OLE 2 should feel that it is a rational and logical "next step" for OLE. OLE 2 builds on the OLE 1 compound document functionality by adding several new features, most noticeably in-place activation. At the same time, the underlying supporting infrastructure has been redesigned to provide a robust platform on which OLE 2 and future application-interoperability products can be built.

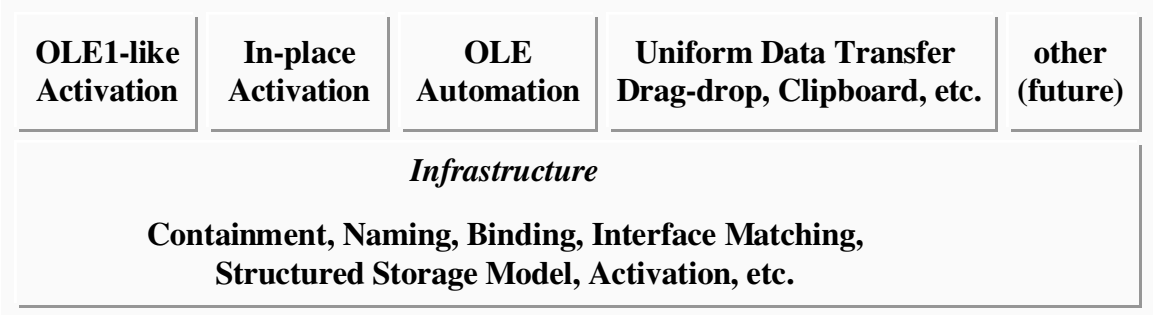
1.1. What is OLE 2?

OLE 2 is primarily a set of standard interfaces by which application integration is achieved. Applications provide and/or use each others' services through these standard interfaces. Most of the implementation of function defined in these interfaces is provided by applications; OLE 2 provides the binding and communication support required to connect the applications.

Some of the main goals of OLE 2 are:

- define a common infrastructure for application integration and extensible interfaces;
- enable programmability of applications;
- enable in-place activating (editing);
- enable drag/drop transfers within and between applications and objects.

Conceptually, the relationship between these areas is as shown below.



As with OLE 1, the design provides for extensibility of the set of interfaces over time, so that applications are not limited by "lowest-common denominator" standards. The infrastructure interfaces and services provide the necessary support for negotiating the set of interfaces that applications have in common. The more application-specific interfaces (the upper layer in the above diagram) should be considered as a starter set. These interfaces will support some level of integration among applications that support these

interfaces; the interfaces do not limit future applications to only these interfaces. This specification includes the definition of what those interfaces include.

1.2. Major Infrastructure Improvements

OLE 2 introduces some architectural changes to the infrastructure of OLE. Some of these changes are for the purpose of making the basic concepts of linking and embedding more generally applicable in practice, more robust, and more efficient. These changes will be perceived by users as removal of constraints, but not really new features.

Another set of infrastructure changes will not be directly perceived by users; these changes relate to making the OLE services easier for application developers to program, by removing unnecessary differences in how applications deal with these services, and removing some complexity of concurrency management and sequencing of operations within an application.

1.2.1. Naming and Links

Consider a monthly financial report, showing a company's sales, outgoing costs and net income for the month, and cumulative income for the financial year to date. Such a report will contain several tables of figures, and probably some charts of the data showing trends, along with explanatory text and other material.

In preparing such a report, the author will want to use a word processor to prepare the text, perhaps from boilerplate material, and one or more spreadsheets that are used to calculate the numbers of interest from the raw data. Tables corresponding to ranges in the spreadsheet will be linked into the word processor document, so that as calculations are updated, the document is changed to reflect the new figures. Similarly, charts embedded in the document are updated from the linked areas of spreadsheet.

OLE 1 (as its name implies) provides support for embedding objects into documents, and for linking objects. In the above scenario, it is useful for the author to embed the spreadsheet data in the document, so that he has a single self-contained document that can be transported, given to reviewers for comment, and so on. Embedding the data allows the derivation of the figures to be audited by those who receive the document in this form.

However, OLE 1 is limited in that links can only be established to obtain data from an object that is itself stored as a file in the file system: if the object is embedded within another document (or the same document!) there is no way for the link client to activate the object, to get an update of data, or open it for editing when the user so requests. So in this case, the author cannot both achieve the link arrangements he needs to get automatic updating of the elements of his report, and the embedding behavior that is so desirable for sharing this document with other people, and transporting it to other places.

To achieve the behavior required in this and other scenarios, OLE 2 introduces an extension to the model of links and the mechanism for naming the referenced object. This extension allows a link to refer to some object that may be embedded somewhere down inside a document. Support for activating the referenced object is included in OLE 2, and in turn requires a modicum of assistance from applications that manage container documents and objects, to activate objects that they contain.

OLE 2 also provides support for tracking links as the source and the destination of the link are moved or copied in the most important common ways.

While this extension to the naming and linking model does not directly provide the end user with more features or functionality, it greatly increases the utility of the OLE 1 functionality, by making it more uniformly applicable, and exploiting the synergy between linking and embedding.

1.2.2. Storage Management

There are many efficiency considerations in dealing with compound documents, especially when multiple applications are involved in creation and management of these documents. As embedding is used for ob-

jects which would otherwise have been implemented as parts of the applications native functionality, it becomes important that OLE should not impose substantial operational overhead on these objects.

One area where OLE 1 introduces some inefficiency is in the storage management and data transfer between container and editor applications. OLE 1 requires that objects be loaded into memory as a whole, and copies objects in the process of transferring them to the editor application. For moderate sizes of objects, the overheads are tolerable, but for large objects, the constraints become severe. Consider an object like a voice or video annotation, which is visually small but requires a large amount of storage. It is unreasonable to require these objects to be linked rather than embedded, just to avoid the additional costs.

OLE 2 introduces a sophisticated storage system for containment hierarchies of objects (such as is required for compound documents). This system permits applications to manage (i.e.: optimize) their transfers between memory and persistent store, so that objects need not be loaded into memory in their entirety. Container applications do not need to load the object's data into memory (except what is needed for drawing pictures etc.), so the data is not copied when it is passed to the editor application. Instead, that application reads and writes directly from the container's persistent storage. The OLE 2 storage system provides separation between the activities of the container and the objects, and provides a transaction model of updating, so that even when an object has made changes to its persistent store, the container may revert to the original state of the document, and another instance of the original may be opened before the changes are committed. The state of the original document reflects the original state of the contained objects.

The OLE 2 storage system supports partial updating of data, and only the changes are stored prior to the commitment of the transaction. This support can be exploited by applications to implement fast incremental save operations, and also to implement full save operations which minimize storage space consumption, with little programming.

1.2.3. More Uniform Object Model

OLE 2 reduces the number of pointlessly different kinds of interface that the developer must deal with just to get basic functionality to work. In particular, simple single object editors deal only with object interfaces, and avoid the complexity of the OLE 1 types OLESERVER, OLESERVERDOC, and OLEOBJECT. At the other end of the scale, fully featured applications which act both as container and containee, and perhaps as link servers, do not need to see separate interfaces for client-side and server-side operations. OLECLIENTDOC and OLESERVERDOC are converged into the notion of a container, and the application can implement this container support just once.

Obviously, as will be described below, OLE 2 is also introducing support for new functionality, which in turn introduces new interfaces for the developer to deal with. Compared to OLE 1, OLE 2 should be thought of as giving the developer a choice between a simpler implementation of the same functionality (as extended by the naming and storage functionality described above), and more functionality for same amount of application complexity. Clearly there will be some cost in converting an OLE 1 application to an OLE 2 application. We have endeavored to make the cost reasonably small, and worthwhile.

1.2.4. Concurrency

The use of two or more applications in managing a single document introduces the potential for concurrent execution, which must be properly managed. In OLE 1, this concurrency was exposed to the application developer to control, in the form of operations on objects whose completion must be awaited, and (on the server side) requiring complex state machines to properly respond to sequences of requests from clients.

OLE 2 uses a more disciplined model of invoking operations on objects, so that a client application need not implement a complex concurrency control mechanism, but can behave more modally while an operation is in progress. There is always some amount of potential for asynchronous behavior while two editors are open, but this is more contained in OLE 2.

On the server side, when operations are requested, more of the information required to respond is made available in the call, and the object states and transitions are simplified, so that the server application has to do less work keeping track of what to expect next, and carrying transient information from one call to the next.

1.3. Major Feature Improvements

OLE 2 introduces mechanisms by which applications can optionally provide and use richer interfaces into applications. These divide into two categories. First is a series of interfaces which support greater integration of the user interface for working with compound documents. In OLE 1, embedded objects are visually contained only while they are inactive, and separated in different windows while they are active. Second is OLE Automation, which potentially enables automation of tasks, and composition of applications into systems that are customized for a particular problem or user task.

1.3.1. In-place Activation

OLE 2 defines interfaces by which applications can cooperate to support the activation of objects in place in the container context, provided that both applications support it. If either does not, then the user is given the OLE 1-style activation in another window. In any case, the user has the option to invoke activation in another window, through the Open verb, for cases where the user requires more display space, control tools, etc. that the object's application cannot provide in-place.

When the user activates an object, the menus and controls change to reflect those of the object. This allows the user to retain a sense of place, and to manipulate the object in its container context which is important for some tasks such as annotation. In addition, there is never confusion of how to get the object “back” into the container, nor complex interactions if the user switches back to the container's window while the object is open. In the most common cases, many of the usability difficulties introduced by OLE 1 will simply not arise.

Since OLE 2 still provides for activation in an new window, the concurrency situations can still occur, and applications must deal with it. However, the major benefit of using in-place interaction for most of the time is that the user will be even less aware of the difference between objects and native data in containers, and the context switching that occurs. The user will be less distracted by artifacts of technology, and more free to concentrate on the task at hand.

1.3.2. Direct Manipulation: Drag/Drop

OLE 2 defines a mechanism and set of API functions for using drag/drop to transfer data between objects, and objects between containers. This will further reduce the intrusion of barriers between applications into the user’s work, since moving text in a document and moving charts between documents will follow the same paradigm.

In addition, OLE 2 provides for the use of pop-up menus for invoking the transfer (clipboard) commands, and invoking verbs on objects. It is anticipated that applications will start to use these menus also for their internal data and editing commands.

1.3.3. OLE Automation: Driving an Application from Outside

OLE 2 provides a means by which an application can define a set of operations with arguments, and make these accessible to other programs. This enables other applications to invoke these operations after binding to the objects. OLE 2 defines an interface for exposing these operations, and for invoking them. This enables application programming languages[1] to allow the user to create programs that operate across

[1] Sometimes called macro languages.

applications as well as within them. Clearly, over time we anticipate that programming tools will permit command aware applications to be combined into custom hyper-applications.

This document does not contain further technical information relating to OLE Automation; such information is found in accompanying documentation. With this exception, the present document provides a comprehensive technical specification of OLE 2.

1.4. What is the Relation Between OLE 1 and OLE 2?

OLE 2 is a compatible re-implementation of OLE 1, enhanced with added functionality. OLE 2 applications will transparently interoperate with OLE 1 applications without having to be aware that the other party is not OLE 2-aware. Of course, only OLE 1 functionality will be available to the user in those cases: old applications will not magically acquire the ability to do in-place interaction.

1.5. What Does It Mean for an Application to Support OLE 2?

To be compliant with OLE 2, applications must support the infrastructure interfaces. Doing only this will not be very interesting. Applications should support one or more of the application integration interfaces to be a useful participant and to provide the end user with the benefits of integration. Application designers must choose which of the capabilities of OLE 2 are important to their customers, and support the interfaces needed to make these capabilities available. As a hypothetical example, a page layout application may want to support only the ability to contain embedded objects, but not the ability for itself be so contained. Further, it may choose to support in-place activation of those objects, or only open activation. Independently, the application may choose to allow itself to be programmatically manipulated with OLE Automation.

1.6. Organization of This Specification

This specification is organized as follows:

Chapter 1. Introduction

> This chapter gives a high level overview of the contents of OLE 2. It is intended to give the reader a quick introduction to the features and improvements introduced in OLE 2.

Chapter 2. User Model and User Interface Guidelines

> This chapter specifies and illustrates the user model and the user interface implementation guidelines for OLE 2. It starts with a walk through of an in-place activation example. This gives the reader a good impression of the feel of in-place activation in a compound document scenario. The remainder of the chapter describes the specifics of the user model of how objects are created, edited, and transferred within documents.

Chapter 3. Architectural Overview

> This chapter gives a comprehensive overview of the components of the OLE 2 architecture. It describes the basic object model and interfaces upon which OLE 2 is built. It gives the reader an overall picture of how the infrastructure pieces of OLE 2 interact and how an OLE 2 application is constructed. It describes the possible states of objects and presents the basics of how objects are created and transferred. This chapter is intended to give the reader an overall picture of what OLE 2 is architecturally; the details of the OLE 2 functions and interfaces follow in the subsequent chapters.

Chapter 4. The Component Object Model

> This chapter presents the basic underlying functions and interfaces used in OLE 2, building on the overview presented in the previous chapter. Detailed function prototypes and descriptions are

presented. The concepts discussed in this chapter are not germane to compound documents *per se*; rather they are applicable over a broader scope.

Chapter 5. Compound Document Interfaces & Functions

This chapter and the next describe the interfaces and functions that are at the heart of providing compound document functionality. That is, in this chapter and the next is found the functionality that supports creating and manipulating the things the user thinks of as "embeddings" and "links."

Chapter 6. Data & Presentation Transfer & Caching

This chapter discusses the mechanisms by which objects are drawn on the screen and the printer or provide data to their containers. Support for obtaining object renderings for more than one target device is discussed. The mechanism by which asynchronous notifications are communicated between objects is presented. The last part of the chapter examines exactly how all these interfaces are pieced together and used by containers, handlers, and servers.

Chapter 7. Linking and Naming

This chapter gives the motivation and design behind the major new naming scheme introduced in OLE 2 – monikers. A moniker is a new referencing mechanism which is used as a persistent handle to refer to an object. It can later be used to bind to the object (the process by which an application loads the object and acquires a real memory pointer to the object). This chapter presents the details of the interfaces and functions used to manipulate monikers.

This chapter also discusses how OLE 2 link objects function.

Chapter 8. Persistent Storage for Objects

This chapter presents the improved persistent storage scheme for OLE 2. OLE 1 required that objects be loaded into memory as a whole. OLE 2 introduces a sophisticated storage system for containment hierarchies of objects (such as is required for compound documents). This system permits applications to manage (i.e. optimize) their transfers between memory and persistent storage, so that objects need not be loaded into memory in their entirety. This chapters defines the interfaces and functions used to manage the persistent storage of objects.

Chapter 9. Drag & Drop and the Clipboard

This chapter discusses the interfaces and APIs with which drag-drop is implemented. It also specifies the interfaces and APIs by which data is transferred through the clipboard in a Cut and Paste operation.

Chapter 10. In-place Activation

OLE 2 provides the ability for an embedded object to be activated within the window of its container. This is known as "in-place activation." Chapter 2 presents the user interface of in-place activation; this chapter describes the functions and interfaces which enable it, the conventions that applications must follow to support it, and how the pieces cooperate with the underlying window system operations.

Chapter 11. Concurrency

OLE 1 exposed a complicated asynchronous concurrency model. OLE 2 uses a more disciplined model of invoking operations on objects, so that a client application need not implement a complex concurrency control mechanism, but can behave more modally while an operation is in progress. This chapter presents the basics of this improved OLE 2 concurrency mechanism.

Chapter 12: Data Formats for Properties and Property Sets

This chapter describes standard data formats for representing property sets. It also defines one standard property set, for Document Summary Information.

Appendix A. System Registration Database Entries

This appendix lists the entries that an application needs to make in the system registration database when installed in the user's system.