OpenDoc Class Reference

For the Mac OS

Addison-Wesley Publishing Company

Reading, Massachusetts Menlo Park, California New York
Don Mills, Ontario Wokingham, England Amsterdam Bonn
Sydney Singapore Tokyo Madrid San Juan
Paris Seoul Milan Mexico City Taipei

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Contents

About This Book

OpenDoc is a set of shared libraries that you can use to build editors and viewers for compound documents and other component software. This book provides reference documentation for OpenDoc on the Mac OS platform. It describes the platform-independent and Mac OS-specific classes, methods, types, constants, and error codes (exceptions) defined by OpenDoc. If you are developing a part editor for the Mac OS platform, you need the information in this book.

This book is a companion to the *OpenDoc Programmer's Guide for the Mac OS,* which provides an architectural overview, synthesizes design concepts, and gives specific programming recommendations to illuminate the information in this book. Before you read this book, you should already be familiar with the basic concepts of OpenDoc, as described in the first chapter of the *Programmer's Guide*. As you read the information in this book, you may also want to refer to the relevant chapters of the *Programmer's Guide* for information on how the classes and methods described here work together in a functioning part editor.

This book does not provide code samples. For detailed tutorial instructions and code listings taken from functioning Mac OS part editors, see the *OpenDoc Cookbook for the Mac OS*.

Organization

Part 1 describes the public OpenDoc classes and methods. Class descriptions are ordered alphabetically. Within each class description, the methods of that class are described in alphabetical order.

Part 2 describes the OpenDoc types and constants, grouped by topic.

The appendix describes requirements for runtime installation of a shell plug-in on the Mac OS platform.

Class Descriptions

Class descriptions in this book are in alphabetical order. Each class description states how the class is to be used—whether the class can be subclassed, how objects of the class are created, and how a part can use an object of the class. If a class is not used directly by parts but only by the OpenDoc document shell or a container application, the class description makes this restriction clear.

Abstract Superclasses

This book and other OpenDoc manuals use the term *abstract superclass* to describe a class, such as `ODPersistentObject` (page 534), that must be subclassed rather than used directly. In some cases, a superclass is not abstract, but implements basic functionality that can be further extended through subclassing. In the case of `ODTransform` (page 736), for example, parts can create and use objects of the superclass and need not implement subclasses.

Inheritance Relationships

The `ODObject` class (page 425) is the root of the OpenDoc class hierarchy. The *OpenDoc Programmer's Guide for the Mac OS* contains illustrations that show the entire OpenDoc class hierarchy. In this book, each class description shows the location of the class in the OpenDoc class hierarchy by listing its ancestors and its subclasses.

The ancestors of a given class are shown as a path up the class hierarchy beginning with the immediate superclass of the class being described and ending with `ODObject`. An arrow points from each ancestor class in the branch to its immediate superclass. For example, the following branch appears in the class description of `ODDocument` (page 130). The branch indicates that `ODRefCntObject` is the superclass of `ODDocument` and `ODObject` is the superclass of `ODRefCntObject`.

Superclasses `ODRefCntObject` → `ODObject`

The subclasses of a given class are shown as a comma-separated list in alphabetical order. For example, the following list appears in the class

description of `ODNameSpace` (page 414); it indicates that `ODNameSpace` has two subclasses, `ODObjectNameSpace` and `ODValueNameSpace`.

Subclasses `ODObjectNameSpace`, `ODValueNameSpace`

Only immediate subclasses are shown; if you are interested in descendant classes at lower levels in the hierarchy, you can refer to the descriptions of the subclasses or to the diagram in the *Programmer's Guide*.

Method Descriptions

For each OpenDoc class, descriptions of its public methods follow the general class description. Methods are described in alphabetical order. If a method is not used directly by parts but only by the OpenDoc document shell or a container application, the method description makes this restriction clear.

Because method names are not necessarily unique, cross-references to method descriptions in this book use a fully qualified method name with the notation *ClassName*::*MethodName*. For example, `ODFacet::CreateCanvas` means the `CreateCanvas` method of the `ODFacet` class.

"Does" Versus "Should"

Most method descriptions in this book state what the method actually does. If a method can be overridden, however, its description states what the override method *should* do. If you override the method, you should implement your override method to behave as the description says it should. If you call such a method, you can assume that the method behaves as it should, but you should be aware that the actual behavior depends on how each override method is implemented.

Object References

OpenDoc objects are always passed by reference when they are used as parameters to methods or as values returned from methods. To emphasize this fact, this book uses the term *reference* exclusively to mean *reference to an object*. The way in which objects are referenced may vary among programming

languages. If you use the C++ language, note that the use of the term *reference* in this book does not necessarily imply a C++ reference.

"This" Object

Because OpenDoc is an object-oriented class library, multiple objects of a given OpenDoc class can be created at runtime. For this reason, method descriptions use the term *this object* (for instance, "this part" or "this frame") to refer to the specific object whose method is being called.

Override Information

If a method must be or cannot be overridden, the method description makes this restriction clear. If an override method must or cannot call its inherited method, the method description makes this restriction clear. Otherwise, the following default information can be assumed for each method where overriding information is not explicitly stated.

If you subclass *ClassName*, you can override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

OpenDoc and SOMobjects

OpenDoc objects follow the System Object Model™ (SOM™), an object-oriented programming technology for building class libraries that support object binding at runtime. On the Mac OS platform, the OpenDoc class libraries are built with SOMobjects for Mac OS, Apple Computer's implementation of SOM.

The interfaces to SOM classes must be written in the CORBA Interface Definition Language (IDL), a programming-language-neutral syntax for creating interfaces. The interfaces are compiled separately from the implementations of the classes by the SOMobjects IDL compiler, which supports object-oriented programming languages such as C++ and procedural programming languages such as C.

Because OpenDoc uses SOMobjects and IDL, part editors and other OpenDoc classes that have been created with different compilers or in different programming languages can nevertheless communicate properly with one another. Furthermore, they can be independently revised and extended and still work together.

IDL Prototypes

Method prototypes in this book are presented in IDL syntax, which is similar to that of C and C++. IDL includes essentially the same character set, whitespace rules, comment styles, preprocessing capabilities, identifier-naming rules, and rules for literals. But there are a few notable differences in source-code appearance when declaring or calling methods of SOM-based objects:

- In IDL method declarations, each parameter declaration is preceded by a *directional attribute* (`in`, `out`, or `inout`) that notes whether the parameter is used as an input, a result, or both.
- The C++ interface to any method of a SOM object includes an extra initial parameter, the environment parameter (`ev`), used by all SOM methods to pass exceptions. See the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS* for information on SOM exception handling.
- The C interface to any SOM method includes another extra parameter (`somSelf`) before the environment parameter, specifying the object to which the method call is directed.

As an example of IDL syntax, here is the prototype for the `AddProperty` method of the `ODStorageUnit` class:

```
ODStorageUnit AddProperty(in ODPropertyName propertyName);
```

The directional attribute `in` indicates that the `propertyName` parameter is only passed into the method.

The SOMobjects IDL compiler converts IDL declarations into declarations and stub definitions in the selected implementation language. It adds any necessary parameters and converts `out` and `inout` parameters appropriately for the selected language. For example, `out` parameters may be implemented as pointers.

SOM Development

All OpenDoc objects are SOM objects, descended from the class `SOMObject`. Your subclass of `ODPart` must likewise be a SOM class. If you want other classes you define to be SOM classes, then you must write your interfaces in IDL, separate from your implementations. You must compile your interfaces with the SOMobjects IDL compiler, which can produce header files and stub implementation source files in the various programming languages supported by the SOMobjects IDL compiler. Options to the compiler specify which files to produce. You complete your development by writing your implementations into the stub implementation files and compiling them, along with the header files, using a standard compiler for your programming language.

For information on SOMobjects, see *SOMobjects for Mac OS*. For information on using the SOMobjects IDL compiler on the Mac OS platform, see the *OpenDoc Cookbook for the Mac OS*. For a more detailed description of the Interface Definition Language and instructions on programming with SOM, see *SOMobjects Developer Toolkit Users Guid*e and *SOMobjects Developer Toolkit Programmers Reference Manual* from IBM.

Conventions Used in This Book

This book uses various conventions to present certain types of information.

Special Fonts

All code listings, reserved words, names of data structures, constants, fields, parameters, and methods are shown in Courier (`this is Courier`).

A term with an OpenDoc-specific meaning is shown in **boldface** when it is first used and defined. The glossary in the *OpenDoc Programmer's Guide for the Mac OS* contains definitions of all such terms.

Types of Notes

There are several types of notes used in this book.

Note
A note formatted like this contains information that is interesting but possibly not essential to an understanding of the main text. The title may be more descriptive than "Note," for example, a note that pertains only to the Mac OS platform may have the title "Mac OS." ◆

IMPORTANT
A note like this contains information that is essential for an understanding of the main text. ▲

Mac OS Information

OpenDoc is a cross-platform technology, and most of its concepts and features are platform-independent. Thus, even though this book is specifically designed for Mac OS developers, the information is organized and presented in as platform-neutral a manner as possible.

Throughout this book, any methods, types, and constants that are specific to the Mac OS are called out as such. That way, you can get a general idea of how platform-specific your code must be, and therefore how simple or complex it may be to convert it to another platform.

Component Integration Laboratories

OpenDoc is presented and maintained through a nonprofit organization devoted to promoting cross-platform standards, architectures, and protocols in a vendor-independent fashion. This organization, Component Integration Laboratories (CI Labs), is composed of a number of platform and application vendors with a common interest in solving OpenDoc issues and promoting interoperability.

CI Labs supports several levels of participation through different membership categories. If you are interested in shaping the future direction of component software, or if you simply need to be kept abreast of the latest developments,

you can become a member. For an information packet, send your mailing address to

Component Integration Laboratories
PO Box 61747
Sunnyvale, CA 94088-1747

Telephone	408-864-0300
FAX	408-864-0380
Internet	cilabs@cilabs.org
World Wide Web	http://www.cilabs.org

PART 1

Classes and Methods

PART 1

Classes and Methods

ODAddressDesc

Superclasses `ODDesc` → `ODObject`

Subclasses none

An object of the `ODAddressDesc` class is a wrapper for an address descriptor structure (type `AEAddressDesc`), a descriptor structure that contains a target destination address.

Description

Every Apple event includes an attribute specifying the destination address of the target application. You specify the destination address using an address descriptor record (a descriptor record of type `AEAddressDesc`). An address descriptor record is simply a descriptor record that contains an application's address.

For more information on Apple events and the `AEAddressDesc` type, see the "Introduction to Apple Events" chapter of *Inside Macintosh: Interapplication Communication*. For general information on scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents a summary description of the `ODAddressDesc` method, followed by a detailed description.

`InitODAddressDesc` Initializes this address descriptor.

InitODAddressDesc

The `InitODAddressDesc` method initializes this address descriptor.

```
void InitODAddressDesc ();
```

DISCUSSION

There is no factory method for the `ODAddressDesc` class; after creating a new address descriptor object, OpenDoc or your part must call this method to initialize the new address descriptor object.

ODAppleEvent

Superclasses `ODRecord` → `ODDescList` → `ODDesc` → `ODObject`

Subclasses none

An object of the `ODAppleEvent` class is a wrapper for an Apple event structure (type `AppleEvent`).

Description

An Apple event structure is a data structure of type `AppleEvent` containing a list of keyword-specified descriptor records that name the attributes and parameters of an event. An Apple event structure is different from an Apple event object.

For more information on Apple event structures and the `AppleEvent` type, see the "Introduction to Apple Events" chapter of *Inside Macintosh: Interapplication Communication*. For general information on scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents a summary description of the `ODAppleEvent` method, followed by a detailed description.

`InitODAppleEvent` Initializes this Apple event structure.

InitODAppleEvent

The `InitODAppleEvent` method initializes this Apple event structure.

```
void InitODAppleEvent ();
```

DISCUSSION

There is no factory method for the `ODAppleEvent` class; after creating a new Apple event structure, OpenDoc or your part must call this method to initialize the new Apple event structure.

ODArbitrator

Superclasses `ODObject`

Subclasses none

An object of the `ODArbitrator` class manages temporary ownership of shared resources or features among parts.

Description

A **focus** is a designation of ownership of a given shared resource. Foci are defined according to the type of resource that each represents. For example, focus types include the keystroke focus, menu bar focus, and selection focus. Other foci can be associated with system-wide resources like serial ports. Foci are owned by frames. The frame's part relinquishes ownership of foci when asked to, such as when another part is requesting ownership or when a frame is deleted.

When a document is opened, the session object creates a single arbitrator object. All parts of that document share the arbitrator object; you can obtain a reference to it by calling the session object's `GetArbitrator` method (page 582).

The OpenDoc arbitrator keeps track of which part owns a focus by consulting the focus module for that focus. A focus is registered if it has a focus module. The arbitrator is used by the OpenDoc dispatcher to determine where to send events. For example, the dispatcher, which is responsible for distributing events to part editors, directs keyboard events to the part that currently owns the keystroke focus and menu events to the part that currently owns the menu bar focus.

Foci may be exclusive or nonexclusive. All of the standard foci defined by OpenDoc are **exclusive**, meaning that only one frame at a time can own a focus. The `ODArbitrator` class also supports **nonexclusive** foci, which can be owned by several frames. In such cases, the arbitrator provides a central

location in which you can record and later obtain a list of the owners for that focus.

For more information related to foci, focus sets, focus types, and focus modules, see the descriptions for the classes `ODFocusModule` (page 258), `ODFocusOwnerIterator` (page 272), `ODFocusSet` (page 279), and `ODFocusSetIterator` (page 284). For more information related to the dispatcher, see the `ODDispatcher` class description (page 108).

Methods

This section presents summary descriptions of the `ODArbitrator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Creating a Focus Set

`CreateFocusSet`	Creates and initializes a focus set.

Focus Transfer

`RequestFocus`	Requests that the ownership of the specified focus be assigned to the specified frame.
`RequestFocusSet`	Requests that the ownership of each focus in the specified focus set be assigned to the specified frame.
`RelinquishFocus`	Called by the current owner of the specified focus to relinquish ownership of it.
`RelinquishFocusSet`	Called by the current owner of each focus in the specified focus set to relinquish ownership of it.
`TransferFocus`	Directly transfers a focus from its current owner to another.
`TransferFocusSet`	Assigns the specified frame as the owner of each focus in the specified focus set.

Registering Focus Modules

`IsFocusRegistered`	Returns a Boolean value that indicates whether the specified focus is registered.
`RegisterFocus`	Adds the specified focus module for the specified focus.

`UnregisterFocus`	Removes the association between the specified focus and the focus module that manages it.
`GetFocusModule`	Returns a reference to a focus module for the specified focus.

Determining Focus Ownership

`IsFocusExclusive`	Returns a Boolean value that indicates whether the specified focus is exclusive.
`AcquireFocusOwner`	Returns a reference to the frame that owns the specified exclusive focus.
`CreateOwnerIterator`	Creates a focus-owner iterator to give callers access to the frames that own a nonexclusive focus.

AcquireFocusOwner

The `AcquireFocusOwner` method returns a reference to the frame that owns the specified exclusive focus.

```
ODFrame AcquireFocusOwner (in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type to be acquired, expressed as a 32-bit value.
return value	A reference to the frame that owns the specified exclusive focus, or `kODNULL` if the focus is not owned by any frame.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

A part can obtain a reference to the owner of a specified exclusive focus by calling this method. This method looks up the focus module for the given focus

and calls that focus module's `AcquireFocusOwner` method. If the focus is not registered, then the focus has no focus module and this method is never called.

This method increments the reference count of the returned frame. When you have finished using that frame, you should call its `Release` method.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODFocusModule::AcquireFocusOwner` method (page 262).
The `ODSession::Tokenize` method (page 598).

CreateFocusSet

The `CreateFocusSet` method creates and initializes a focus set.

```
ODFocusSet CreateFocusSet ();
```

return value A reference to a new focus set object.

DISCUSSION

Your part calls this method to create a focus set to pass to the arbitrator's `RequestFocusSet` method.

EXCEPTIONS

`kODErrOutofMemory`	There is not enough memory to allocate the focus set object.

SEE ALSO

The `ODArbitrator::RequestFocusSet` method (page 54).
The `ODFocusSet` class (page 279).

CreateOwnerIterator

The `CreateOwnerIterator` method creates a focus-owner iterator to give callers access to the frames that own a nonexclusive focus.

```
ODFocusOwnerIterator CreateOwnerIterator (
                                    in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type whose owners you want to access, expressed as a 32-bit value.
return value	A reference to a new focus-owner iterator object, or `kODNULL` if the specified focus is exclusive.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

To get access to the owner of an exclusive focus, use the arbitrator's `AcquireFocusOwner` method.

While you are using a focus-owner iterator, you should not modify the list of focus owners. You must postpone adding items to or removing items from the list of focus owners until after you have deleted the iterator.

EXCEPTIONS

`kODErrFocusNotRegistered`	The specified focus is not registered.
`kODErrOutOfMemory`	There is not enough memory to allocate the focus-owner iterator object.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).

The `ODArbitrator::AcquireFocusOwner` method (page 45).
The `ODSession::Tokenize` method (page 598).
The `ODFocusOwnerIterator` class (page 272).

GetFocusModule

The `GetFocusModule` method returns a reference to a focus module for the specified focus.

```
ODFocusModule GetFocusModule (in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type to be acquired, expressed as a 32-bit value.
return value	A reference to a focus module for the specified focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

IsFocusExclusive

The `IsFocusExclusive` method returns a Boolean value that indicates whether the specified focus is exclusive.

```
ODBoolean IsFocusExclusive (in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type to be tested, expressed as a 32-bit value.
return value	`kODTrue` if the specified focus is exclusive, otherwise `kODFalse`.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

EXCEPTIONS

`kODErrFocusNotRegistered`
The specified focus is not registered.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

IsFocusRegistered

The `IsFocusRegistered` method returns a Boolean value that indicates whether the specified focus is registered.

```
ODBoolean IsFocusRegistered (in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type to be tested, expressed as a 32-bit value.
return value	`kODTrue` if the specified focus is registered, otherwise `kODFalse`.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

A focus is registered if it has a focus module.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

RegisterFocus

The `RegisterFocus` method adds the specified focus module for the specified focus.

```
void RegisterFocus (in ODTypeToken focus,
                    in ODFocusModule focusModule);
```

`focus` A tokenized string representing the focus type to be assigned to the specified focus module, expressed as a 32-bit value.

`focusModule`
A reference to a focus module that is to manage the specified focus, or `kODNULL` to create a standard exclusive focus module.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

EXCEPTIONS

`kODErrFocusAlreadyRegistered`
The specified focus is already registered.

`kODErrOutOfMemory` There is not enough memory to allocate the default focus module.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

RelinquishFocus

The `RelinquishFocus` method is called by the current owner of the specified focus to relinquish ownership of it.

```
void RelinquishFocus (in ODTypeToken focus,
                      in ODFrame relinquishingFrame);
```

`focus` A tokenized string representing the focus type to be relinquished, expressed as a 32-bit value.

`relinquishingFrame`
A reference to a frame relinquishing ownership of the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

If the focus is exclusive, it has no owner after this method executes successfully.

EXCEPTIONS

`kODErrFocusNotRegistered`
The specified focus is not registered.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODArbitrator::RelinquishFocusSet` method (page 52).
The `ODSession::Tokenize` method (page 598).

RelinquishFocusSet

The `RelinquishFocusSet` method is called by the current owner of each focus in the specified focus set to relinquish ownership of it.

```
void RelinquishFocusSet (in ODFocusSet focusSet,
                         in ODFrame relinquishingFrame);
```

`focusSet` A reference to a focus set containing the foci to be relinquished.

`relinquishingFrame`
A reference to a frame relinquishing ownership of each focus in the specified focus set.

EXCEPTIONS

`kODErrFocusNotRegistered`
One of the specified foci is not registered.

SEE ALSO

The `ODArbitrator::RelinquishFocus` method (page 51).

RequestFocus

The `RequestFocus` method requests that the ownership of the specified focus be assigned to the specified frame.

```
ODBoolean RequestFocus (in ODTypeToken focus,
                        in ODFrame requestingFrame);
```

`focus` A tokenized string representing the focus type whose ownership is being requested, expressed as a 32-bit value.

`requestingFrame`
A reference to a frame requesting the focus.

return value `kODTrue` if the frame obtained the focus, otherwise `kODFalse`.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

Your part calls this method to request a single focus for one of its frames; to request multiple foci, your part calls the arbitrator's `RequestFocusSet` method. The return value indicates whether the specified frame obtained ownership of the specified focus.

If the specified focus is nonexclusive, the specified frame is automatically granted ownership of the focus. If it is exclusive, the focus module calls the `BeginRelinquishFocus` method of the current owner's part to see if the current owner is willing to give it up.

If the request is granted, the new ownership relationship is stored in the relevant focus modules. If the request fails, the existing ownership relationships remain intact.

If the request is granted, the arbitrator contains a reference to the frame. Parts should relinquish the focus in the `DisplayFrameClosed` or `DisplayFrameRemoved` methods.

EXCEPTIONS

`kODErrFocusNotRegistered`
The specified focus is not registered.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODArbitrator::RequestFocusSet` method (page 54).
The `ODPart::BeginRelinquishFocus` method (page 467).
The `ODSession::Tokenize` method (page 598).

RequestFocusSet

The `RequestFocusSet` method requests that the ownership of each focus in the specified focus set be assigned to the specified frame.

```
ODBoolean RequestFocusSet (in ODFocusSet focusSet,
                           in ODFrame requestingFrame);
```

`focusSet` A reference to a focus set being requested.

`requestingFrame`
A reference to a frame requesting the focus.

return value `kODTrue` if the frame obtained the focus set, otherwise `kODFalse`.

DISCUSSION

For the requested ownership to be granted, all existing owners of exclusive foci within the specified set must be willing to relinquish these same foci.

If all of the foci, exclusive and nonexclusive, are attainable, then the request is granted. However, if even one focus is unattainable, ownership of the focus set is not granted.

If the request is granted, the new ownership relationships are stored in the relevant focus modules. If the request fails, the existing ownership relationships remain intact.

If the request is granted, the arbitrator contains a reference to the frame. Parts should relinquish the focus in the `DisplayFrameClosed` or `DisplayFrameRemoved` methods.

EXCEPTIONS

`kODErrFocusNotRegistered`
One of the specified foci is not registered.

SEE ALSO

The `ODArbitrator::RequestFocus` method (page 53).

TransferFocus

The `TransferFocus` method directly transfers a focus from its current owner to another.

```
void TransferFocus (in ODTypeToken focus,
                    in ODFrame transferringFrame,
                    in ODFrame newOwner);
```

`focus` A tokenized string representing the focus type to be transferred, expressed as a 32-bit value.

`transferringFrame`
A reference to a frame that is transferring ownership of the focus. This frame need not be the current owner.

`newOwner` A reference to a frame that is to be the new focus owner.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

Your part can call this method, instead of methods that relinquish a focus, to restore ownership to a previous owner. For example, when a part requests the modal focus, as it does when displaying a modal dialog box, that part should save the previous owner of the modal focus, and transfer ownership back to the previous owner when the dialog box is dismissed. This technique may be necessary if modal dialog boxes can be nested.

This method calls the new owner's `FocusAcquired` method if the new owner is not the transferring frame. If the previous owner is not the transferring frame, OpenDoc also calls the previous owner's `FocusLost` method.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).

The `ODArbitrator::TransferFocusSet` method (page 57).
The `ODPart::FocusAcquired` method (page 498).
The `ODPart::FocusLost` method (page 499).
The `ODSession::Tokenize` method (page 598).

TransferFocusSet

The `TransferFocusSet` method assigns the specified frame as the owner of each focus in the specified focus set.

```
void TransferFocusSet (in ODFocusSet focusSet,
                       in ODFrame transferringFrame,
                       in ODFrame newOwner);
```

`focusSet` A reference to a focus set being transferred.

`transferringFrame`
A reference to a frame that is transferring ownership of the focus set.

`newOwner` A reference to a frame that is to be the new focus set owner.

DISCUSSION

This method calls the new owner's `FocusAcquired` method if the new owner is not the transferring frame. If the previous owner is not the transferring frame, OpenDoc also calls the previous owner's `FocusLost` method.

SEE ALSO

The `ODArbitrator::TransferFocus` method (page 56).
The `ODPart::FocusAcquired` method (page 498).
The `ODPart::FocusLost` method (page 499).

UnregisterFocus

The `UnregisterFocus` method removes the association between the specified focus and the focus module that manages it.

```
void UnregisterFocus (in ODTypeToken focus);
```

`focus` A tokenized string representing the focus type to be removed, expressed as a 32-bit value.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

EXCEPTIONS

`kODErrFocusNotRegistered`
The specified focus is not registered.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

ODBinding

Superclasses `ODObject`

Subclasses none

An object of the `ODBinding` class represents the OpenDoc binding object that performs the runtime binding of part editors to the parts in a document.

Description

When a document is opened, the session object creates a single binding object. All parts of that document share the binding object; the document shell or a container application can obtain a reference to it by calling the session object's `GetBinding` method (page 583).

OpenDoc binds part editors to part data when a part is read in or when its part editor is changed. The binding object gathers information provided by the installed part editors, preferences specified by the user, and part-kind information stored with parts. It uses that information to choose an editor for each part in the document. For more information about binding, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents a summary description of the `ODBinding` method, followed by a detailed description.

`ChooseEditorForPart`

Chooses the editor to be used to edit the specified part.

ChooseEditorForPart

The `ChooseEditorForPart` method chooses the editor to be used to edit the specified part.

```
ODEditor ChooseEditorForPart (in ODStorageUnit thePartSU,
                              in ODType newKind);
```

`thePartSU`	A reference to the part's storage unit.
`newKind`	If the specified part is a new part being created, its part kind; otherwise `kODNULL`.
return value	An opaque platform-specific value specifying the chosen part editor.

DISCUSSION

This method is called by the container suite. The document shell, container applications, and parts cannot call this method.

This method chooses the appropriate editor for the specified part based on the editors that are installed on the machine and the part kind of the specified part. For a new part that is being created, the specified storage unit is empty and the `newKind` parameter specifies the part kind. For a part that exists, the storage unit contains the part kind(s).

SEE ALSO

The `ODEditor` type (page 899).
The `ODType` type (page 846).

ODCanvas

Superclasses `ODObject`

Subclasses none

An object of the `ODCanvas` class is a wrapper for a graphics-system-specific drawing structure that represents a drawing environment—the environment for constructing an image.

Description

A canvas object represents a drawing environment—the destination for drawing calls—for any of the available graphics systems that your part uses. Your part uses the standard platform drawing calls for your graphics system to render its content on a canvas. A canvas can refer to anything a graphics system knows how to draw into; for example, a graphics port, print job, offscreen pixel buffer, bitmap, structured display list, or stream of PostScript™ code. A graphics-system-specific drawing structure may retain state information (for example, pen color) that influences how the drawing calls are interpreted.

Your part creates a canvas object by calling the window-state object's `CreateCanvas` method (page 827). To create a canvas to attach to a particular facet, your part can call that facet's `CreateCanvas` method (page 232).

Each canvas object includes one or more graphics-system-specific drawing structures, which are not deleted when the canvas is released. If you create a canvas, you must separately create the underlying drawing structure, and you are responsible for deleting that drawing structure when the canvas is released.

For more information on the graphics systems available on the Mac OS platform, see *Inside Macintosh: Imaging with QuickDraw* and *Inside Macintosh: QuickDraw GX Graphics*.

Canvas Characteristics

A canvas must be either dynamic or static:

- A dynamic canvas is a drawing canvas that is interactive. Windows, which can be scrolled or paged to display different portions of a part's content, are good examples of dynamic canvases.
- A static canvas is a drawing canvas that cannot be changed once it is rendered; for instance, one that cannot be scrolled. A printed image (or onscreen print preview) is an example of a static canvas because the user cannot scroll elements on the page or otherwise interact with it once it is drawn.

Your part can display parts differently based on this distinction. For example, you might use scroll bars on the screen but not on printed material. You might also use different drawing calls for printing than for screen display because printing varies from platform to platform.

A canvas is also defined as being either onscreen or offscreen:

- An onscreen canvas is the main canvas of the window or print job.
- An offscreen canvas is used to improve performance by allowing you to draw a complex image to an offscreen cache, and then quickly transfer the completed image to the onscreen canvas, with full freedom to alter or distort the image in the process. For instance, you can create an offscreen canvas to do double-buffering or image manipulation (such as changing the tinting or translucency of an image).

When a part creates a canvas it specifies, for the lifetime of the canvas, whether the canvas is dynamic or static and whether it is onscreen or offscreen.

A canvas is further defined in terms of a transformation matrix and its coordinate bias:

- A transformation matrix called the **bias transform** describes the transform that is applied to measurements in a canvas's coordinate space to change them into standard platform-normal coordinates.

  Bias transforms are used to negotiate between the containing part and the embedded part. The negotiation occurs when several canvases, each defined in its own coordinate space, are combined into a single coordinate system. Facets use bias transforms to convert geometry from one coordinate space to the other, usually so that a part can use the geometry to display on its canvas. Each canvas is scaled, rotated, and translated by redefinition of its coordinates in the bias canvas's coordinate space. Thus, once you set up your offscreen canvas for drawing in your own coordinate system, you can also use it to make sure that all point, frame, and facet geometry is properly converted for you.

- The **coordinate bias**, defined by the bias transform, is the difference between the canvas's coordinate space and the standard platform-normal coordinate space. The coordinate bias usually takes the form of an offset in the origin, a change in the polarity of one or more axes, and possibly a change in scale.

Offscreen Imaging

Canvases can be attached to individual facets. If a particular facet in a window's facet hierarchy has an attached canvas, it and all its embedded facets (and their embedded facets, and so on) draw to that canvas. Each facet inherits its canvas from its containing facet; the inherited canvas is called the **parent canvas**. For most drawing, only a window's root facet needs a canvas.

If a particular part needs an offscreen canvas, however, it can attach a canvas to one of its facets on a display frame. The canvas has a reference to that facet and also a reference to the part that created the canvas and attached it to a facet—the **owning part**. The reference enables the canvas to notify the owning part that its content has changed and that it needs to be updated. The owning part is responsible for copying the image of the offscreen canvas to its parent canvas during updates.

The owning part of a canvas need not be the same as its facet's part. For instance, a containing part may customize the drawing of an embedded part by assigning the facet of the embedded part to an offscreen canvas. In this case,

the containing part must make itself the owning part so that it can control the drawing of the facet on the screen.

For added convenience, offscreen canvases maintain updating information that mirrors their onscreen equivalents. This lets embedded parts interact with their drawing environment in a consistent manner, whether the parts are displayed in the window canvas or in an offscreen canvas.

Methods

This section presents summary descriptions of the `ODCanvas` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [M] are specific to the Mac OS platform.

Canvas Characteristics

`AcquireBiasTransform`
: Returns a reference to the bias transform associated with this canvas.

`SetBiasTransform`
: Assigns the specified bias transform to this canvas.

`GetFacet`
: Returns a reference to the facet associated with this canvas.

`SetFacet`
: Assigns the specified facet to this canvas.

`AcquireOwner`
: Returns a reference to the part that owns this canvas.

`SetOwner`
: Assigns the specified owning part to this canvas.

`IsDynamic`
: Returns a Boolean value that indicates whether this canvas is dynamic.

`IsOffscreen`
: Returns a Boolean value that indicates whether this canvas is offscreen.

Drawing Structures

`GetPlatformCanvas`
: Returns the drawing structure for the specified graphics system for this canvas.

`SetPlatformCanvas`
: Assigns the drawing structure for the specified graphics system to this canvas.

`HasPlatformCanvas`
: Returns a Boolean value that indicates whether this canvas has or can generate a drawing structure for the specified graphics system.

`GetGXViewPort` [M]	Returns the QuickDraw GX view port drawing structure for this canvas.
`GetQDPort` [M]	Returns the QuickDraw graphics-port drawing structure for this canvas.

Invalidating and Updating

`AcquireUpdateShape`	Returns a reference to the shape object defining the area of this canvas that needs to be updated.
`ResetUpdateShape`	Sets the update shape for this canvas to the empty shape.
`Invalidate`	Adds the specified area to the update shape for this canvas, ensuring that the specified area of this canvas is updated.
`Validate`	Subtracts the specified area from the update shape for this canvas.

Printing

`GetPlatformPrintJob`	Returns the print job for the specified graphics system for this canvas.
`SetPlatformPrintJob`	Assigns the print job for the specified graphics system to this canvas.
`HasPlatformPrintJob`	Returns a Boolean value that indicates whether this canvas has a print job for the specified graphics system.

AcquireBiasTransform

The `AcquireBiasTransform` method returns a reference to the bias transform associated with this canvas.

```
ODTransform AcquireBiasTransform ();
```

return value A reference to the bias transform associated with this canvas, or `kODNULL` if no bias transform has previously been assigned to this canvas.

DISCUSSION

If you call methods that modify the returned transform, you must then call this canvas's `SetBiasTransform` method to set its bias transform to the modified transform.

This method increments the reference count of the returned transform. When you have finished using that transform, you should call its `Release` method.

SEE ALSO

The `ODCanvas::SetBiasTransform` method (page 76).

AcquireOwner

The `AcquireOwner` method returns a reference to the part that owns this canvas.

```
ODPart AcquireOwner ();
```

return value A reference to the part that owns this canvas, or `kODNULL` if the part does not exist.

DISCUSSION

This method increments the reference count of the returned part. When you have finished using that part, you should call its `Release` method.

SEE ALSO

The `ODCanvas::SetOwner` method (page 77).

AcquireUpdateShape

The `AcquireUpdateShape` method returns a reference to the shape object defining the area of this canvas that needs to be updated.

```
ODShape AcquireUpdateShape ();
```

return value A reference to the shape object defining the area of this canvas that needs to be updated.

DISCUSSION

OpenDoc calls this method internally; your part modifies the update shape of a canvas by calling its facet's `Invalidate` and `Validate` methods.

This method increments the reference count of the returned shape. When the caller has finished using that shape, it should call the shape's `Release` method.

SEE ALSO

The `ODCanvas::Invalidate` method (page 74).
The `ODCanvas::ResetUpdateShape` method (page 75).
The `ODCanvas::Validate` method (page 80).
The `ODFacet::Invalidate` method (page 245).
The `ODFacet::Validate` method (page 252).

GetFacet

The `GetFacet` method returns a reference to the facet associated with this canvas.

```
ODFacet GetFacet ();
```

return value A reference to the facet associated with this canvas.

SEE ALSO

The `ODCanvas::SetFacet` method (page 77).

GetGXViewport

Mac OS

The `GetGXViewport` method returns the QuickDraw GX view port drawing structure for this canvas.

```
ODPlatformCanvas GetGXViewport ();
```

return value A 32-bit value identifying the QuickDraw GX view port object for this canvas. You must cast the return value to type `gxViewPort`.

DISCUSSION

If you are using QuickDraw GX for imaging, you can call this method in your part's `Draw` method or whenever you need to draw anything into a facet.

If this canvas has a QuickDraw GX view port drawing structure and QuickDraw GX is installed, this method returns that view port. If this canvas has a QuickDraw window drawing structure, this method generates a new QuickDraw GX view port for this canvas using the QuickDraw drawing structure; the method then returns the new QuickDraw GX view port.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
This implementation of OpenDoc does not support QuickDraw GX, QuickDraw GX is not installed or available, or this canvas has no QuickDraw GX or QuickDraw drawing structure.

SEE ALSO

The `ODCanvas::GetQDPort` method (page 71).

GetPlatformCanvas

The `GetPlatformCanvas` method returns the drawing structure for the specified graphics system for this canvas.

```
ODPlatformCanvas GetPlatformCanvas (in ODGraphicsSystem g);
```

`g`	A 16-bit value specifying the graphics system you want to use for this canvas. Valid values for `g` are platform dependent.
return value	A 32-bit value identifying the graphics-system-specific drawing structure for this canvas. Before using the return value, you must cast it to a valid graphics-system type (such as `GrafPtr` for QuickDraw or `gxViewPort` for QuickDraw GX).

DISCUSSION

You call this method to get the graphics-system-specific drawing structure (for instance, a window or view port) when you need to draw into a facet.

On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`). You must specify the graphics system because, on some platforms, a canvas can have drawing structures for two or more graphics systems simultaneously.

If this canvas has a drawing structure for the specified graphics system, this method returns that drawing structure.

On the Mac OS platform, this method generates and returns a drawing structure if the following conditions are all true:

- QuickDraw GX is installed.
- You specify the QuickDraw GX graphics system.
- This canvas does not have a QuickDraw GX drawing structure, but does have a QuickDraw window drawing structure.

In that situation, this method uses the QuickDraw drawing structure to generate a new QuickDraw GX drawing structure for this canvas.

If this canvas does not have and cannot generate a drawing structure for the specified graphics system, this method returns `kODNULL`.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
: This implementation of OpenDoc does not support the specified graphics system, that graphics system is not installed or available, or this canvas has no drawing structure for that graphics system.

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODCanvas::HasPlatformCanvas` method (page 72).
The `ODCanvas::SetPlatformCanvas` method (page 78).
For more information on graphics-system-specific drawing structures, see the documentation for your graphics system.

GetPlatformPrintJob

The `GetPlatformPrintJob` method returns the print job for the specified graphics system for this canvas.

```
ODPlatformPrintJob GetPlatformPrintJob (
                                    in ODGraphicsSystem g);
```

`g`
: A 16-bit value specifying the graphics system you want to use for this canvas. Valid graphics systems are platform dependent.

return value
: A 32-bit value identifying the graphics-system-specific print job for this canvas. Before using the return value, you must cast it to a valid print job type (such as `THPrint` for QuickDraw or `gxJob` for QuickDraw GX).

DISCUSSION

On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`).

If this canvas has a print job for the specified graphics system, this method returns that print job. On the Mac OS platform, the return value may be used to determine whether you are printing to a PostScript printer. You need to call this method only when you are creating a static canvas to use as a print job.

A canvas can have only one print job (for one graphics system) even if it has drawing structures for more than one graphics system.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
: This implementation of OpenDoc does not support the specified graphics system, that graphics system is not installed or available, or this canvas does not have a print job for that graphics system.

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODCanvas::HasPlatformPrintJob` method (page 73).
The `ODCanvas::SetPlatformPrintJob` method (page 79).

GetQDPort

Mac OS

The `GetQDPort` method returns the QuickDraw graphics-port drawing structure for this canvas.

```
GrafPtr GetQDPort ();
```

return value A pointer to a QuickDraw graphics port.

DISCUSSION

If you are using QuickDraw for imaging, you call this method in your part's `Draw` method or whenever you need to draw anything into a facet.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
: This implementation of OpenDoc does not support QuickDraw, QuickDraw is not installed or available, or this canvas has no QuickDraw drawing structure.

SEE ALSO

The `ODCanvas::GetGXViewport` method (page 68).

HasPlatformCanvas

The `HasPlatformCanvas` method returns a Boolean value that indicates whether this canvas has or can generate a drawing structure for the specified graphics system.

```
ODBoolean HasPlatformCanvas (in ODGraphicsSystem g);
```

`g`
: A 16-bit value specifying the graphics system you want to check for this canvas. Valid graphics systems are platform dependent.

return value
: `kODTrue` if this canvas has or can generate a drawing structure for the specified graphics system, otherwise `kODFalse`.

DISCUSSION

On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`).

If this canvas has a drawing structure for the specified graphics system, this method returns true.

On the Mac OS platform, if the following conditions are all true, this method returns true because a canvas can generate a QuickDraw GX drawing structure from a QuickDraw window drawing structure:

- QuickDraw GX is installed.
- You specify the QuickDraw GX graphics system.
- This canvas does not have a QuickDraw GX drawing structure, but does have a QuickDraw window drawing structure.

If this canvas does not have and cannot generate a drawing structure for the specified graphics system, this method returns false.

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODCanvas::GetPlatformCanvas` method (page 69).
The `ODCanvas::SetPlatformCanvas` method (page 78).

HasPlatformPrintJob

The `HasPlatformPrintJob` method returns a Boolean value that indicates whether this canvas has a print job for the specified graphics system.

```
ODBoolean HasPlatformPrintJob (in ODGraphicsSystem g);
```

`g`	A 16-bit value specifying the graphics system you want to check for this canvas. Valid graphics systems are platform dependent.
return value	`kODTrue` if this canvas has a print job for the specified graphics system, otherwise `kODFalse`.

DISCUSSION

On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`).

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODCanvas::GetPlatformPrintJob` method (page 70).
The `ODCanvas::SetPlatformPrintJob` method (page 79).

Invalidate

The `Invalidate` method adds the specified area to the update shape for this canvas, ensuring that the specified area of this canvas is updated.

```
void Invalidate (in ODShape shape);
```

`shape` A reference to the shape object defining the area to add to the update shape for this canvas.

DISCUSSION

OpenDoc calls this method internally to mark an area of a canvas that needs updating. Your part typically calls its facet's `Invalidate` method instead of this method because that method transforms and clips the shape from the coordinate space of the facet to the coordinate space of its canvas.

SEE ALSO

The `ODCanvas::Validate` method (page 80).
The `ODFacet::Invalidate` method (page 245).
The `ODFrame::Invalidate` method (page 317).

IsDynamic

The `IsDynamic` method returns a Boolean value that indicates whether this canvas is dynamic.

```
ODBoolean IsDynamic ();
```

return value `kODTrue` if this canvas is dynamic, otherwise `kODFalse`.

DISCUSSION

The dynamic or static characteristic of a canvas is set when the canvas is created and cannot be changed.

IsOffscreen

The `IsOffscreen` method returns a Boolean value that indicates whether this canvas is offscreen.

```
ODBoolean IsOffscreen ();
```

return value `kODTrue` if this canvas is offscreen, otherwise `kODFalse`.

DISCUSSION

The onscreen or offscreen characteristic of a canvas is set when the canvas is created and cannot be changed.

ResetUpdateShape

The `ResetUpdateShape` method sets the update shape for this canvas to the empty shape.

```
void ResetUpdateShape ();
```

DISCUSSION

OpenDoc calls this method internally while processing update events and after all the invalidated canvases have been updated.

SEE ALSO

The `ODCanvas::AcquireUpdateShape` method (page 67).

SetBiasTransform

The `SetBiasTransform` method assigns the specified bias transform to this canvas.

```
void SetBiasTransform (in ODTransform x);
```

`x` A reference to the bias transform to assign to this canvas.

DISCUSSION

The bias transform is calculated by concatenating the internal transform (if any) of this canvas's owning frame with the internal transform of the root frame. You can use this method to add a vertical flip and offset between frames of graphics systems whose coordinate spaces have different handedness; for example, Mac OS- and Windows-based frames on an OS/2 system.

After this method executes successfully, any preexisting bias transform is released and this canvas owns the new transform. You can call the `AcquireBiasTransform` method of this canvas to obtain a reference to the resulting bias transform.

You should release the transform object `x` immediately after passing it as a parameter to this method, without using or modifying it further.

SEE ALSO

The `ODCanvas::AcquireBiasTransform` method (page 65).

SetFacet

The `SetFacet` method assigns the specified facet to this canvas.

```
void SetFacet (in ODFacet facet);
```

`facet` A reference to the facet to assign to this canvas.

DISCUSSION

OpenDoc calls this method when the canvas is added to the facet.

SEE ALSO

The `ODCanvas::GetFacet` method (page 67).

SetOwner

The `SetOwner` method assigns the specified owning part to this canvas.

```
void SetOwner (in ODPart owner);
```

`owner` A reference to the owning part to assign to this canvas.

DISCUSSION

You typically call this method immediately after your part creates a custom canvas.

SEE ALSO

The `ODCanvas::AcquireOwner` method (page 66).

SetPlatformCanvas

The `SetPlatformCanvas` method assigns the drawing structure for the specified graphics system to this canvas.

```
void SetPlatformCanvas (in ODGraphicsSystem g,
                        in ODPlatformCanvas c);
```

`g` A 16-bit value specifying the graphics system whose drawing structure you are setting. Valid graphics systems are platform dependent.

`c` A 32-bit value identifying the graphics-system-specific drawing structure to assign to this canvas, or `kODNULL` to remove the drawing structure for a graphics system. Valid values for `c` are graphics-system-dependent.

DISCUSSION

On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`). For QuickDraw, the platform canvas should be a QuickDraw graphics port (type `GrafPtr`); for QuickDraw GX, it should be a QuickDraw GX view port object (type `gxViewPort`).

You can assign any graphics-system-specific drawing structure that a part might use. On some platforms, a canvas can have drawing structures for two or more graphics systems simultaneously.

EXCEPTIONS

`kODErrInvalidGraphicsSystem` This implementation of OpenDoc does not support the specified graphics system or that graphics system is not installed or available.

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODCanvas::GetPlatformCanvas` method (page 69).
The `ODCanvas::HasPlatformCanvas` method (page 72).

SetPlatformPrintJob

The `SetPlatformPrintJob` method assigns the print job for the specified graphics system to this canvas.

```
void SetPlatformPrintJob (in ODGraphicsSystem g,
                          in ODPlatformPrintJob j);
```

`g` A 16-bit value specifying the graphics system whose print job you are setting. Valid graphics systems are platform dependent.

`j` A 32-bit value identifying the graphics-system-specific print job to assign to this canvas, or `kODNULL` to clear the print job. Valid values for `j` are graphics-system-dependent.

DISCUSSION

You need to call this method only when you are creating a static canvas to use as a print job.

On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`). For QuickDraw, the platform print job should be a QuickDraw print job (type `THPrint`); for QuickDraw GX, it should be a QuickDraw GX print job (type `gxJob`).

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODCanvas::GetPlatformPrintJob` method (page 70).
The `ODCanvas::HasPlatformPrintJob` method (page 73).

Validate

The `Validate` method subtracts the specified area from the update shape for this canvas.

```
void Validate (in ODShape shape);
```

`shape` A reference to the shape object defining the area to subtract from the update shape for this canvas.

DISCUSSION

OpenDoc calls this method internally to mark an area of a canvas that no longer needs updating. Your part typically calls its facet's `Validate` method instead of this method because that method transforms and clips the shape from the coordinate space of the facet to the coordinate space of its canvas.

SEE ALSO

The `ODCanvas::Invalidate` method (page 74).
The `ODFacet::Validate` method (page 252).
The `ODFrame::Validate` method (page 332).

ODClipboard

Superclasses `ODObject`

Subclasses none

An object of the `ODClipboard` class provides data-transfer services between OpenDoc documents and their parts and between OpenDoc and non-OpenDoc documents.

Description

On each platform, OpenDoc provides an implementation of the `ODClipboard` class that gives a common, platform-independent clipboard interface for all parts. Each such implementation of `ODClipboard` provides access to a platform-specific clipboard (also known as the **system clipboard**) via OpenDoc storage units. This approach not only shields part editors from the underlying system clipboard mechanism, it also allows for more complex data transfers between different parts and documents, including linking.

When a document is opened, the session object creates a single clipboard object. All parts of that document share the clipboard object; you can obtain a reference to it by calling the session object's `GetClipboard` method (page 583). You must not cache this object; instead, you must call the session object's `GetClipboard` method whenever you need the clipboard object.

Parts typically invoke `ODClipboard` methods in response to Edit menu commands such as Copy, Cut, Paste, and Paste As and during certain linking and drag-and-drop operations. In addition to the traditional data-transfer features associated with these commands, OpenDoc's ability to clone objects allows a data-transfer operation to involve not only intrinsic content of the source part, but also the content of embedded parts, which may be of any part kind and embedded to any depth. The OpenDoc clipboard mechanism also allows more flexibility in how items are pasted. For example, the transferred data can be embedded or incorporated into the destination part with optional translations.

You should access the clipboard only when your part is running in the frontmost process and your part owns the clipboard focus. You can acquire the clipboard focus by calling the arbitrator object's `RequestFocus` method (page 53). Acquiring the clipboard focus ensures that no other part can access or modify the data while your transfers are taking place. An active part must therefore acquire the clipboard focus before enabling the appropriate commands in the Edit menu. The recommended strategy is to acquire focus in your part's `AdjustMenus` method (page 465) and to relinquish focus in your part's `HandleEvent` method (page 506).

Once your part has the clipboard focus, you can write data to, or read data from, the clipboard. If you are writing data to the clipboard, you must first call the `Clear` method (page 87) of the clipboard object. Data transfers requested by any part operate on the clipboard object's content storage unit. You obtain a reference to that storage unit by calling the clipboard's `GetContentStorageUnit` method (page 90). The content storage unit contains the data most recently cut or copied to the clipboard. You must not cache that storage unit; instead you must call the `GetContentStorageUnit` method whenever you need to access the storage unit.

To transfer persistent objects to or from the clipboard, you must clone them using the cloning methods of their draft object. Additional methods of `ODClipboard` transfer data between the system clipboard and the clipboard object's storage unit.

For efficient transfer of large amounts of data to and from the clipboard, OpenDoc supports **promises**. The source part can delay a data transfer by writing a promise to the clipboard. If, and only if, a destination subsequently seeks to retrieve the data, the source part fulfills its promise by writing the data to the clipboard. This time-saving technique is transparent to the destination part.

The `GetUpdateID` method (page 91) of the clipboard object returns an update ID that uniquely identifies the current generation or version of the clipboard. You can save this update ID and use it to detect subsequent changes in the clipboard content.

Parts that support Cut and Paste must support undo of those operations. If the user moves objects from one part to another and then undoes the move, the objects must be reinstated at the source. This can happen only if the part initiating the cut and the part performing the paste both support undo. Your part should notify the clipboard object whenever a Cut, Copy, or Paste operation is done, undone, or redone by calling the clipboard object's

`ActionDone` (page 84), `ActionUndone` (page 86), or `ActionRedone` (page 85) methods, respectively.

For more information about cloning, promises, and using the clipboard object, see the chapter on data transfer in the *OpenDoc Programmer's Guide for the Mac OS*. For more information on cloning methods, see the descriptions of the classes `ODDraft` (page 145) and `ODPart` (page 445). For information on undoing and redoing actions, see the chapter on windows and menus in the *OpenDoc Programmer's Guide for the Mac OS* and the `ODUndo` class description (page 779). For information on using a link specification to indicate that the source part can create a link, see the `ODLinkSpec` class description (page 379).

Methods

This section presents summary descriptions of the `ODClipboard` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are called only by the document shell or container applications. Methods marked [M] are specific to the Mac OS platform.

Accessing

`Clear` Immediately removes all the data stored in this clipboard object.

`GetContentStorageUnit` Returns a reference to the storage unit containing this clipboard object's current content.

Update Control

`ExportClipboard` [D] Updates the system clipboard if this clipboard object has been changed since the last update of the system clipboard.

`DraftClosing` [D] Notifies this clipboard object that the specified draft is about to be closed or reverted.

`DraftSaved` [D] Notifies this clipboard object that the specified draft has been saved.

`GetUpdateID` Returns an update ID identifying the current generation or version of this clipboard object.

`SetPlatformClipboard`
: Copies any data of the specified types from this clipboard object to the system clipboard if this clipboard object has changed since the last update to the system clipboard.

Dialog Control

`ShowPasteAsDialog` [M]
: Displays the Paste As dialog box and sets the appropriate dialog items according to the input parameters.

Undoing Clipboard Operations

`ActionDone`
: Notifies this clipboard object that a Cut, Copy, or Paste action was done.

`ActionUndone`
: Notifies this clipboard object that a Cut, Copy, or Paste action was undone.

`ActionRedone`
: Notifies this clipboard object that a Cut, Copy, or Paste action was redone.

ActionDone

The `ActionDone` method is called to notify this clipboard object that a Cut, Copy, or Paste action was done.

```
ODUpdateID ActionDone (in ODCloneKind cloneKind);
```

`cloneKind`
: The kind of clone operation that copied data to or from the clipboard. The clone kind must be one of Cut (`kODCloneCut`), Copy (`kODCloneCopy`), or Paste (`kODClonePaste`).

return value
: The update ID identifying the version of the clipboard content involved with this operation.

DISCUSSION

Your part should call this method whenever it performs a Cut, Copy, or Paste operation. You should cache the returned update ID in case you ever need to undo or redo the action.

EXCEPTIONS

`kODErrIllegalClipboardCloneKind`
The specified clone kind was not `kODCloneCopy`, `kODCloneCut`, or `kODClonePaste`.

SEE ALSO

The `ODClipboard::ActionRedone` method (page 85).
The `ODClipboard::ActionUndone` method (page 86).
The `ODPart::RedoAction` method (page 518).
The `ODPart::UndoAction` method (page 528).
The `ODUndo` class (page 779).

ActionRedone

The `ActionRedone` method is called to notify this clipboard object that a Cut, Copy, or Paste action was redone.

```
void ActionRedone (in ODUpdateID updateID,
                   in ODCloneKind originalCloneKind);
```

`updateID` The update ID identifying the version of the clipboard content involved with this operation.

`originalCloneKind`
The type of clone operation that copied data to or from the clipboard. The clone kind must be one of Cut (`kODCloneCut`), Copy (`kODCloneCopy`), or Paste (`kODClonePaste`).

DISCUSSION

Your part should call this method whenever it redoes a Cut, Copy, or Paste operation. The update ID should be the ID returned by the `ActionDone` method at the time your part originally performed the action. The clone kind should be the same clone kind as specified in that call to the `ActionDone` method.

EXCEPTIONS

`kODErrIllegalClipboardCloneKind`
: The specified clone kind was not `kODCloneCopy`, `kODCloneCut`, or `kODClonePaste`.

SEE ALSO

The `ODClipboard::ActionDone` method (page 84).
The `ODClipboard::ActionUndone` method (page 86).
The `ODPart::RedoAction` method (page 518).

ActionUndone

The `ActionUndone` method is called to notify this clipboard object that a Cut, Copy, or Paste action was undone.

```
void ActionUndone (in ODUpdateID updateID,
                   in ODCloneKind originalCloneKind);
```

`updateID` The update ID identifying the version of the clipboard content involved with this operation.

`originalCloneKind`
: The type of clone operation that copied data to or from the clipboard. The clone kind must be one of Cut (`kODCloneCut`), Copy (`kODCloneCopy`), or Paste (`kODClonePaste`).

DISCUSSION

Your part should call this method whenever it undoes a Cut, Copy, or Paste operation. The update ID should be the ID returned by the `ActionDone` method at the time your part originally performed the action. The clone kind should be the same clone kind as specified in that call to the `ActionDone` method.

EXCEPTIONS

`kODErrIllegalClipboardCloneKind`
: The specified clone kind was not `kODCloneCopy`, `kODCloneCut`, or `kODClonePaste`.

SEE ALSO

The `ODClipboard::ActionDone` method (page 84).
The `ODClipboard::ActionRedone` method (page 85).
The `ODPart::UndoAction` method (page 528).

Clear

The `Clear` method immediately removes all data stored in this clipboard object.

```
void Clear ();
```

DISCUSSION

For thread-safe operation, the active part must acquire the clipboard focus before invoking this method. This method also causes the system clipboard to be cleared, but at a time that is dependent on both the platform and the document shell.

The object calling this method must not be holding a storage-unit reference returned by a prior call to the `GetContentStorageUnit` method.

After this method executes successfully, the next call to the `GetContentStorageUnit` method returns a storage-unit object with no properties.

EXCEPTIONS

`kODErrBackgroundClipboardClear`
This clipboard belongs to a background process.

SEE ALSO

The `ODArbitrator::RequestFocus` method (page 53).
The `ODClipboard::GetContentStorageUnit` method (page 90).

DraftClosing

Document shell

The `DraftClosing` method is called to notify this clipboard object that the specified draft is about to be closed or reverted.

```
void DraftClosing (in ODDraft draft);
```

`draft` A reference to the draft being closed or reverted.

DISCUSSION

The document shell or container application calls this method when the specified draft is about to be closed or reverted. Parts must not call this method.

This method can be used to ensure the integrity of this clipboard object's reference to the destination draft object involved in paste operations. If the specified draft is closed or reverted, the clipboard's reference to the draft becomes invalid and must be deleted to avoid the possible corruption of destination draft objects.

DraftSaved

Document shell

The `DraftSaved` method is called to notify this clipboard object that the specified draft has been saved.

```
void DraftSaved (in ODDraft draft);
```

`draft` A reference to the draft that was saved.

DISCUSSION

The document shell calls this method after saving every draft. Parts must not call this method.

Any objects cut from the saved draft that are still on the clipboard must be treated as if they were copied if they are later pasted back into the saved draft.

ExportClipboard

Document shell

The `ExportClipboard` method updates the system clipboard if this clipboard object has been changed since the last update of the system clipboard.

```
void ExportClipboard ();
```

DISCUSSION

OpenDoc calls this method internally. Parts must not call this method, but can instead call the `SetPlatformClipboard` method.

After this method executes successfully, the system clipboard contains the most recently cut or copied items from this clipboard object.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to complete the operation.

This method may also throw platform-specific exceptions.

SEE ALSO

The `ODClipboard::SetPlatformClipboard` method (page 92).

GetContentStorageUnit

The `GetContentStorageUnit` method returns a reference to the storage unit containing this clipboard object's current content.

```
ODStorageUnit GetContentStorageUnit ();
```

return value	A reference to the storage-unit object containing the content of the clipboard.

DISCUSSION

You can read data from or write data to the returned storage unit. You must not cache the returned storage unit; instead you must call this method whenever you need to access the content storage unit.

You must acquire the clipboard focus before calling this method, and relinquish the focus after the data transfer. If you are writing data, you must also call the `Clear` method before calling this method.

The clipboard object handles the creation and destruction of its content storage unit, so you must neither dispose of nor release the returned storage unit.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to complete the operation.

SEE ALSO

The `ODClipboard::Clear` method (page 87).
The `ODClipboard::GetUpdateID` method (page 91).
The `ODStorageUnit` class (page 641).

GetUpdateID

The `GetUpdateID` method returns an update ID identifying the current generation or version of this clipboard object.

```
ODUpdateID GetUpdateID ();
```

return value The update ID of the current clipboard content.

DISCUSSION

Your part may call this method to check whether another part has changed the clipboard. The returned update ID uniquely identifies the current generation or version of the clipboard. Note that update ID values have no significance other than in the context of testing them for equality. Also, the update ID value returned by this method is valid only during the current session.

It is possible, but dangerous, to call this method before your part has acquired the clipboard focus.

IMPORTANT

Clipboard update IDs are used for a different purpose than are link and link-source update IDs. You should never set the update ID of a link-source object to an update ID returned by this method. ▲

SEE ALSO

The `ODClipboard::ActionDone` method (page 84).
The `ODLink::GetUpdateID` method (page 344).
The `ODLinkSource::GetUpdateID` method (page 371).

SetPlatformClipboard

The `SetPlatformClipboard` method copies any data of the specified types from this clipboard object to the system clipboard if this clipboard object has changed since the last update to the system clipboard.

```
void SetPlatformClipboard (in ODPlatformTypeList typeList);
```

`typeList` A reference to the list of platform-dependent types to be copied to the system clipboard.

DISCUSSION

You must call this method prior to invoking any platform-specific service that uses the system clipboard.

This method examines the value type of each value in the contents property of this clipboard's storage unit. If the value type is in the specified platform type-list or if the `typeList` parameter is `kODNULL`, this method copies the data in that value to the system clipboard.

To keep system clipboard behavior consistent, this method does not attempt translation to a specified type that is not present in this clipboard object.

EXCEPTIONS

`kODErrOutOfMemory` There is not enough memory to complete the operation.

This method may also throw platform-specific exceptions.

SEE ALSO

The `ODClipboard::ExportClipboard` method (page 89).
The `ODPart::AdjustMenus` method (page 465).

ShowPasteAsDialog

Mac OS

The `ShowPasteAsDialog` method displays the Paste As dialog box and sets the appropriate dialog items according to the input parameters.

```
ODBoolean ShowPasteAsDialog (
                    in ODBoolean canPasteLink,
                    in ODPasteAsMergeSetting mergeSetting,
                    in ODFacet facet,
                    in ODTypeToken viewType,
                    out ODPasteAsResult theResult);
```

`canPasteLink`
: `kODTrue` if the destination part allows a link to be created, otherwise `kODFalse`.

`mergeSetting`
: A value indicating whether embedding and merging are supported; determines the initial setting for the At the Destination radio buttons.

`facet`
: A reference to the facet from which the Paste As dialog box is triggered.

`viewType`
: A tokenized string representing the initial setting for the view type of the embedded part (if embedding is chosen).

`theResult`
: A structure reflecting the user's selections in the Paste As dialog box.

return value
: `kODTrue` if the user clicked OK to leave the Paste As dialog box, otherwise `kODFalse`.

DISCUSSION

You should call this method in your part's `HandleEvent` method to display the Paste As dialog box when the user selects Paste As from the Edit menu.

If the `canPasteLink` parameter is true, and if the content storage unit contains a link specification and the draft permissions allow writing, then the Paste with Link checkbox is checked.

The `mergeSetting` parameter specifies which At the Destination radio button (Merge with Contents or Embed As) is initially selected in the Paste As dialog box and whether the other button is available. It must be one of the following:

- Embed As is initially selected; Merge with Contents is available (`kODPasteAsEmbed`).
- Embed As is selected; Merge with Contents is disabled (`kODPasteAsEmbedOnly`).
- Merge with Contents is initially selected; Embed As is available (`kODPasteAsMerge`).
- Merge with Contents is selected; Embed As is disabled (`kODPasteAsMergeOnly`).

The `viewType` parameter must be the tokenized form of one of the view-type constants (`kODViewAsFrame`, `kODViewAsLargeIcon`, `kODViewAsSmallIcon`, or `kODViewAsThumbnail`). You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

If the user clicks OK, this method returns true and sets the fields of the output parameter, `theResult`, to indicate the selections the user made in the Paste As dialog box. You must dispose of the non-null `selectedKind`, `translateKind`, and `editor` fields of the `theResult` structure when you are finished using them.

If the user cancels the dialog box, this method returns false and you do not need to take any further action.

EXCEPTIONS

`kODErrNullFacetInput`	The `facet` parameter is null.
`kODErrNullPasteAsResultInput`	The `theResult` parameter is null.
`kODErrOutOfMemory`	There is not enough memory to complete the operation.

SEE ALSO

The `ODPasteAsResult` type (page 888).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

ODContainer

Superclasses `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODContainer` class represents a physical container storing a collection of OpenDoc documents. The `ODContainer` class lets you access these documents independently of the container's physical characteristics or internal structure.

Description

The `ODContainer` class is implemented differently for different platforms and storage mechanisms.

The `ODContainer` class manipulates physical containers. Similarly, the `ODDocument` class (page 130) manipulates documents; the `ODDraft` class (page 145) manipulates drafts; and the `ODStorageUnit` class (page 641) manipulates storage units. This set of related classes, `ODContainer`, `ODDocument`, `ODDraft`, and `ODStorageUnit` is called a **container suite**. Container suite classes are implemented as an integrated set for each platform and storage mechanism because they work intimately with one another at many levels. The container suite used by default on the Mac OS platform is the Bento container suite.

Each container object can contain one or more document objects. Each document object, in turn, can contain one or more draft objects, and each draft object can contain one or more storage-unit objects. Each container, document, draft, and storage unit has with a unique ID.

Only the OpenDoc storage system directly creates container objects. The document shell or container application creates or accesses a container object by calling the storage-system object's `CreateContainer` (page 637) or `AcquireContainer` method (page 635). The storage system ensures that there is only one container object associated with each physical container.

You can access a document of a container by calling the container's `AcquireDocument` method (page 98). The `ODContainer` class is responsible for ensuring that there is only one document object associated with each document in the container.

For more information on how `ODContainer` and other container-suite classes are used, see the chapters on storage and OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODContainer` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are typically called by the document shell or container applications.

Object Retrieval

`AcquireDocument` [D]	Returns a reference to the document object associated with the specified document ID.
`GetStorageSystem`	Returns a reference to the storage-system object that created this container object.

Inquiry

`GetID`	Returns the container ID of this container object.
`GetName`	Returns the name of this container object.

Naming

`SetName` [D]	Sets the name of this container object.

AcquireDocument

Document shell

The `AcquireDocument` method returns a reference to the document object associated with the specified document ID.

```
ODDocument AcquireDocument (in ODDocumentID id);
```

`id`	The ID of the requested document.
return value	A reference to the specified document object.

DISCUSSION

The document shell or container application calls this method when the user opens an OpenDoc document.

This method looks in this container object for the document object associated with the specified document ID. If such a document object exists, the object is returned. If the target document object does not exist, it is created, initialized, and returned.

This method increments the reference count of the returned document object. When the caller has finished using that document, it should call the document's `Release` method.

EXCEPTIONS

`kODErrDocumentDoesNotExists`
The specified document does not exist.

SEE ALSO

The `ODDocumentID` type (page 872).
The `ODRefCntObject::Release` method (page 555).
The `ODDocument` class (page 130).

GetID

The `GetID` method returns the container ID of this container object.

```
ODContainerID GetID ();
```

return value A container ID whose buffer contains data identifying this container object.

DISCUSSION

Although parts can call this method, they usually do not need to know the IDs of their associated containers.

The structure of the data identifying this container depends on the type of container, which was specified when the container was created. For example, the identifier for a Bento file container specifies a file-system file; the identifier for a Bento memory container is a handle for a relocatable memory block.

SEE ALSO

The `ODContainerID` type (page 870).

GetName

The `GetName` method returns the name of this container object.

```
ODContainerName GetName ();
```

return value The name of this container, or an empty sequence if the container does not have a name.

DISCUSSION

Although parts can call this method, they usually do not need to know the names of their associated containers.

If this container has a name, this method returns a copy of that name. Otherwise, the `text` field of the result `ODContainerName` structure represents an empty sequence of characters; that is, the `text` field is an `ODByteArray` structure whose `_length` field contains 0 and whose `_buffer` field contains a null pointer.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODContainerName` type (page 870).
The `ODContainer::SetName` method (page 100).

GetStorageSystem

The `GetStorageSystem` method returns a reference to the storage-system object that created this container object.

```
ODStorageSystem GetStorageSystem ();
```

return value A reference to the storage-system object that created this container object.

SEE ALSO

The `ODStorageSystem` class (page 634).

SetName

Document shell

The `SetName` method sets the name of this container object.

```
void SetName (in ODContainerName name);
```

`name` The new name for this container.

DISCUSSION

The document shell or container application calls this method to set the name of this container.

SEE ALSO

The `ODContainerName` type (page 870).
The `ODContainer::GetName` method (page 99).

ODDesc

Superclasses `ODObject`

Subclasses `ODOSLToken`, `ODObjectSpec`, `ODAddressDesc`, and `ODDescList`

An object of the `ODDesc` class is a wrapper for a descriptor structure (type `AEDesc`), the basic structure used for building Apple event attributes and parameters.

Description

An Apple event descriptor structure consists of a handle to data and a descriptor type that identifies the type of data to which the handle refers. An object of the `ODDesc` class stores the actual data. To use the Apple Event Manager functions on an `ODDesc` object, you must first extract the data from it and then create a descriptor structure from that data.

The `ODDesc` class provides methods for placing and extracting the Apple event descriptor it contains. To convert from an `ODDesc` descriptor object to type `AEDesc`, you may copy the raw data and the descriptor type using the utility routines ODDescToAEDesc and AEDescToODDesc.

For more information on Apple events and the `AEDesc` type, see the "Introduction to Apple Events" chapter of *Inside Macintosh: Interapplication Communication*. For general information on scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODDesc` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Initializing

`InitODDesc`	Initializes this descriptor object.

Accessing Descriptor Data

`GetDescType`	Returns the descriptor type of this descriptor object.
`SetDescType`	Sets the descriptor type of this descriptor object.
`GetRawData`	Returns the raw data contained in this descriptor object.
`SetRawData`	Assigns the specified raw data to this descriptor object; any previous data is deleted.

GetDescType

The `GetDescType` method returns the descriptor type of this descriptor object.

```
ODDescType GetDescType ();
```

return value The descriptor type of this descriptor object.

DISCUSSION

On the Mac OS platform, the descriptor type is defined as type `DescType`; `ODDescType` is a wrapper for that type.

If a descriptor object represents an Apple-event-specific descriptor, the returned data may not be interpretable without using the Apple Event Manager functions.

SEE ALSO

The `ODDescType` type (page 895).

GetRawData

The `GetRawData` method returns the raw data contained in this descriptor object.

```
ODByteArray GetRawData ();
```

return value A byte array whose buffer contains the raw data to be retrieved.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to allocate the byte array structure's buffer.

SEE ALSO

The `ODByteArray` type (page 847).

InitODDesc

The `InitODDesc` method initializes this descriptor object.

```
void InitODDesc ();
```

DISCUSSION

There is no factory method for the `ODDesc` class; after creating a new descriptor object, OpenDoc or your part must call this method to initialize the new descriptor object.

SetDescType

The `SetDescType` method sets the descriptor type of this descriptor object.

```
void SetDescType (in ODDescType descType);
```

`descType` The descriptor type of this descriptor object.

DISCUSSION

On the Mac OS platform, the descriptor type is defined as type `DescType`; `ODDescType` is a wrapper for that type.

This method may change the interpretation of the data in the descriptor object, but it does not change the raw data itself.

SEE ALSO

The `ODDescType` type (page 895).

SetRawData

The `SetRawData` method assigns the specified raw data to this descriptor object; any previous data is deleted.

```
void SetRawData (in ODByteArray data);
```

`data` A byte array whose buffer contains the raw data to be stored.

SEE ALSO

The `ODByteArray` type (page 847).

ODDescList

Superclasses `ODDesc` → `ODObject`

Subclasses `ODRecord`

An object of the `ODDescList` class is a wrapper for a descriptor list (type `AEDescList`), a descriptor structure that is a list of other descriptor structures.

Description

A descriptor list is a data structure of type `AEDescList` defined by the data type `AEDesc`—that is, a descriptor list is a descriptor record whose data handle refers to a list of other descriptor records (unless it is an empty list).

For more information on Apple events and the `AEDescList` type, see the "Introduction to Apple Events" chapter of *Inside Macintosh: Interapplication Communication*. For general information on scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents a summary description of the `ODDescList` method, followed by a detailed description.

`InitODDescList` Initializes this descriptor list.

InitODDescList

The `InitODDescList` method initializes this descriptor list.

```
void InitODDescList ();
```

DISCUSSION

There is no factory method for the `ODDescList` class; after creating a new descriptor list, OpenDoc or your part must call this method to initialize the new descriptor list.

ODDispatcher

Superclasses `ODObject`

Subclasses none

An object of the `ODDispatcher` class is responsible for distributing events to part editors.

Description

When a document is opened, the session object creates a single dispatcher object. All parts of that document share the dispatcher object; you can obtain a reference to it by calling the session object's `GetDispatcher` method (page 584).

The dispatcher maintains a dispatch module dictionary, indexed by event type. The dictionary contains at least one internal dispatch module to handle standard events—such as mouse clicks, keystrokes, and menu commands—of a particular platform. You can extend the OpenDoc dispatching system by installing additional dispatch modules to dispatch new types of events or messages to your part editor. For more information, see the `ODDispatchModule` class description (page 125).

Part editors do not receive events directly from the operating system; rather, OpenDoc notifies the appropriate part editor when an event occurs. To do so, the document shell's event loop calls the dispatcher's `Dispatch` method (page 113) to dispatch events to part editors. The dispatcher handles the events it recognizes by using dispatch module objects to dispatch specific events to individual parts. The dispatcher locates the dispatch module for the specified event in its dispatch module dictionary and calls the dispatch module's `Dispatch` method (page 127). The `Dispatch` method in turn calls the part editor's `HandleEvent` method (page 506) to give the part the opportunity to handle the specified event. If your part contains embedded frames, OpenDoc can also send your part editor special mouse events that occur within and on the borders of your part's embedded frames' facets. The dispatcher leaves all other events, as well as events dispatched to parts but not handled by them, to

the document shell to handle. Events not handled by the document shell are ignored. For more information on event handling in OpenDoc, see the chapter on user events in the *OpenDoc Programmer's Guide for the Mac OS*.

Event Monitoring

OpenDoc allows you to monitor the event stream without interfering with it. By registering a dispatch module as a **monitor** for a specified event type, OpenDoc notifies the dispatch module when an event of that type occurs. You might use a monitor in a debugging environment to monitor events and display a log of the events in a window. In general, you should install monitors and not patch the dispatch module. Patching occurs when the creator of the new dispatch module saves the existing dispatch module and then installs its own replacement dispatch module that calls the original.

Scheduling Idle Time

On platforms with the concept of idle time, such as the Mac OS platform, OpenDoc permits your part to receive idle-time events, also called null events. Your part must register either itself or each frame that is to receive null events by calling the dispatcher's `RegisterIdle` method (page 119).

Mac OS Mouse Regions

Whenever the user moves the mouse, your part editor is responsible for providing feedback to the user that corresponds to the location of the cursor on the screen. For example, most part editors set the cursor to the I-beam when the cursor is inside a text-editing part of a document and change the cursor to an arrow when the cursor is inside the scroll bar of a document. Your part editor can achieve this by setting a mouse region and handling mouse-enter, mouse-within, and mouse-leave events. Those events occur when the user moves the cursor outside the mouse region.

The mouse region defaults to a 1-by-1 pixel area at the mouse location, but can be set larger using the dispatcher's `SetMouseRegion` method (page 122). Your part editor should recalculate the mouse region when it receives a mouse-enter, mouse-within, or mouse-leave event, or it will continue to receive these events as long as the user moves the mouse outside the original mouse region.

Methods

This section presents summary descriptions of the `ODDispatcher` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are called only by the document shell or container applications. Methods marked [M] are specific to the Mac OS platform.

Dispatch Module Manipulation

`AddDispatchModule` Adds the specified dispatch module for the specified event type to the dispatch module dictionary.

`GetDispatchModule` Returns a reference to the dispatch module for the specified event type.

`RemoveDispatchModule` Removes the dispatch module for the specified event type from the dispatch module dictionary.

Event Dispatching

`Dispatch` [D] Dispatches the specified event to the appropriate part.

`Redispatch` Redispatches the specified event to the appropriate part.

Event Monitoring

`AddMonitor` Adds the specified dispatch module, which is a monitor for the specified event type, to the dispatch module dictionary.

`RemoveMonitor` Removes the specified dispatch module, which is a monitor for the specified event type, from the dispatch module dictionary.

Cursor Manipulation

`GetMouseRegion` [D] [M] Returns the current mouse region used in dispatching mouse-enter, mouse-within, and mouse-leave events.

`SetMouseRegion` [M] Assigns the mouse region used in dispatching mouse-enter, mouse-within, and mouse-leave events.

`InvalidateFacetUnderMouse` [M]
Sets the facet pointer, corresponding to the facet under the mouse, to `kODNULL`.

Scheduling Idle Time

`RegisterIdle` [M] Adds the specified part and frame to the list of parts interested in receiving idle time.

`UnregisterIdle` [M] Removes the specified part and frame from the list of parts interested in receiving idle time.

`GetSleepTime` [D] [M] Computes the amount of time the document shell can sleep before it needs to wake up to give idle time to registered frames.

`SetIdleFrequency` [M]
Sets the idle frequency for the specified part or frame.

Scheduling Processor Time

`Yield` [M] Gives processor time to other parts.

Document Shell Termination

`Exit` [D] Sets a Boolean value in the dispatcher that specifies that the document shell should terminate.

`ShouldExit` [D] Returns a Boolean value that indicates whether the document shell should terminate.

AddDispatchModule

The `AddDispatchModule` method adds the specified dispatch module for the specified event type to the dispatch module dictionary.

```
void AddDispatchModule (
                    in ODEventType eventType,
                    in ODDispatchModule dispatchModule);
```

`eventType` A platform-specific event code that specifies the type of event to be handled by the new dispatch module. On the Mac OS platform, the event code is defined as an unsigned 16-bit value.

`dispatchModule`
A reference to a dispatch module to be added.

DISCUSSION

Your part editor calls this method to install a custom dispatch module. This method is not called by part editors under normal circumstances.

EXCEPTIONS

`kODErrIllegalNullDispatchModuleInput`
The `dispatchModule` parameter is null.

SEE ALSO

The `ODEventType` type (page 862).
The `ODDispatcher::GetDispatchModule` method (page 115).
The `ODDispatcher::RemoveDispatchModule` method (page 119).
The `ODDispatchModule` class (page 125).

AddMonitor

The `AddMonitor` method adds the specified dispatch module, which is a monitor for the specified event type, to the dispatch module dictionary.

```
void AddMonitor (in ODEventType eventType,
                 in ODDispatchModule dispatchModule);
```

`eventType` A platform-specific event code that specifies the type of event to be handled by the new dispatch module. On the Mac OS platform, the event code is defined as an unsigned 16-bit value.

`dispatchModule`
A reference to a dispatch module to be added.

DISCUSSION

Your part editor calls this method to install a custom dispatch module as a monitor for the specified event type. This method is not called by part editors under normal circumstances.

EXCEPTIONS

`kODErrIllegalNullDispatchModuleInput`
The `dispatchModule` parameter is null.

SEE ALSO

The `ODEventType` type (page 862).
The `ODDispatcher::RemoveMonitor` method (page 120).
The `ODDispatchModule` class (page 125).

Dispatch

Document shell

The `Dispatch` method dispatches the specified event to the appropriate part.

```
ODBoolean Dispatch (inout ODEventData eventData);
```

`eventData` A platform-specific structure representing an event. On return, the fields of the structure may have been modified. On the Mac OS platform, the structure is defined as a Mac OS event record.

return value `kODTrue` if the event was handled by a part, or `kODFalse` if the event is not associated with an existing dispatch module (or if the event was not handled).

DISCUSSION

The document shell and container applications call this method when an event occurs. This method looks up the dispatch module for the specified event in its

dispatch module dictionary and then calls the dispatch module's `Dispatch` method. On the Mac OS platform, this method may also be called by parts that handle events in dialog event filter routines.

SEE ALSO

The `ODEventData` type (page 860).
The `ODEventInfo` type (page 861).
The `ODDispatchModule::Dispatch` method (page 127).
The `ODPart::HandleEvent` method (page 506).

Exit

Document shell

The `Exit` method sets a Boolean value in the dispatcher that specifies that the document shell should terminate.

```
void Exit ();
```

DISCUSSION

This method is not called by most parts.

SEE ALSO

The `ODDispatcher::ShouldExit` method (page 122).

GetDispatchModule

The `GetDispatchModule` method returns a reference to the dispatch module for the specified event type.

```
ODDispatchModule GetDispatchModule (
                                    in ODEventType eventType);
```

`eventType` A platform-specific event code that specifies the type of event handled by the requested dispatch module. On the Mac OS platform, the event code is defined as an unsigned 16-bit value.

return value A reference to the dispatch module for the specified event type, or `kODNULL` if the event type is not associated with an existing dispatch module.

DISCUSSION

Though not highly recommended, this method might be called to patch a dispatch module. Patching occurs when the creator of the new dispatch module saves the existing dispatch module and then installs its own replacement dispatch module that calls the original. Your part editor calls this method to save the existing dispatch module so it can delegate to that dispatch module when necessary; this method is not called by most parts.

SEE ALSO

The `ODEventType` type (page 862).
The `ODDispatcher::AddDispatchModule` method (page 111).
The `ODDispatcher::RemoveDispatchModule` method (page 119).

GetMouseRegion

Document shell
Mac OS

The `GetMouseRegion` method returns the current mouse region used in dispatching mouse-enter, mouse-within, and mouse-leave events.

```
ODRgnHandle GetMouseRegion ();
```

return value The current mouse region, expressed in QuickDraw global coordinates.

DISCUSSION

The document shell and container applications call this method to get the mouse region that the dispatcher uses to report mouse-enter, mouse-within, or mouse-leave events.

If your part does not want to use the default region (1-by-1 pixel), it should call the dispatcher's `SetMouseRegion` method from its routine for handling mouse-enter, mouse-within, or mouse-leave events.

SEE ALSO

The `ODRgnHandle` type (page 854).
The `ODDispatcher::SetMouseRegion` method (page 122).

GetSleepTime

Document shell
Mac OS

The `GetSleepTime` method computes the amount of time the document shell can sleep before it needs to wake up to give idle time to registered frames.

```
ODSLong GetSleepTime ();
```

return value The computed amount of sleep time, expressed in ticks (60ths of a second).

InvalidateFacetUnderMouse

Mac OS

The `InvalidateFacetUnderMouse` method sets the facet pointer, corresponding to the facet under the mouse, to `kODNULL`.

```
void InvalidateFacetUnderMouse ();
```

DISCUSSION

When a facet is removed, OpenDoc calls this method to set the facet pointer to `kODNULL` (that is, invalidate the cache) and the mouse region to an empty region. This method sets up the `GetMouseRegion` method (on its next call) to recompute which facet is under the mouse and send mouse-enter, mouse-within, or mouse-leave events, as required.

Your part can call this method to reset the mouse region to a 1-by-1 pixel area at the cursor position.

SEE ALSO

The `ODDispatcher::GetMouseRegion` method (page 116).

Redispatch

The `Redispatch` method redispatches the specified event to the appropriate part.

```
ODBoolean Redispatch (inout ODEventData eventData,
                      inout ODEventInfo eventInfo);
```

`eventData`	A platform-specific structure representing an event. On return, the fields of the structure may have been modified. On the Mac OS platform, the structure is defined as a Mac OS event record.
`eventInfo`	A platform-specific structure that contains additional event information. On return, the relevant fields of the structure are filled in if the event was handled.
return value	`kODTrue` if the event was handled by a part, or `kODFalse` if the event is not associated with an existing part (or if the event was not handled).

DISCUSSION

The `eventInfo` structure contains the following information:

- a reference to an embedded frame and an embedded facet of the part (for event types `kODEvtMouseDownEmbedded`, `kODEvtMouseUpEmbedded`, or `kODEvtMouseDownBorder`)
- an `ODPoint` value describing the location of the event, expressed in frame coordinates
- a Boolean value that indicates whether an embedded part propagated an event to its containing part

OpenDoc calls this method when it translates an event. For example, when the standard dispatch module transforms a mouse-down event in the menu bar to a menu event, the dispatch module redispatches the event so that monitors and patches can intercept the new event.

SEE ALSO

The `ODEventData` type (page 860).
The `ODEventInfo` type (page 861).
The `ODDispatcher::Dispatch` method (page 113).
The `ODDispatchModule::Dispatch` method (page 127).

RegisterIdle

Mac OS

The `RegisterIdle` method adds the specified part and frame to the list of parts interested in receiving idle time.

```
void RegisterIdle (in ODPart part,
                   in ODFrame frame,
                   in ODIdleFrequency frequency);
```

`part`	A reference to a part that is to receive null events.
`frame`	A reference to a frame that is to receive null events, or `kODNULL` if the part as a whole is to receive idle time.
`frequency`	The idle frequency, expressed in ticks (60ths of a second).

DISCUSSION

Your part typically calls this method, if it needs idle time, when a facet or frame is added.

SEE ALSO

The `ODIdleFrequency` type (page 862).
The `ODDispatcher::SetIdleFrequency` method (page 121).
The `ODDispatcher::UnregisterIdle` method (page 123).

RemoveDispatchModule

The `RemoveDispatchModule` method removes the dispatch module for the specified event type from the dispatch module dictionary.

```
void RemoveDispatchModule (in ODEventType eventType);
```

`eventType` A platform-specific event code that specifies the type of event to be handled by the new dispatch module. On the Mac OS platform, the event code is defined as an unsigned 16-bit value.

DISCUSSION

This method is not called by part editors under normal circumstances; however, your part editor may call this method to remove a custom dispatch module.

SEE ALSO

The `ODEventType` type (page 862).
The `ODDispatcher::AddDispatchModule` method (page 111).
The `ODDispatcher::GetDispatchModule` method (page 115).

RemoveMonitor

The `RemoveMonitor` method removes the specified dispatch module, which is a monitor for the specified event type, from the dispatch module dictionary.

```
void RemoveMonitor (in ODEventType eventType,
                    in ODDispatchModule dispatchModule);
```

`eventType` A platform-specific event code that specifies the type of event to be removed from the new dispatch module. On the Mac OS platform, the event code is defined as an unsigned 16-bit value.

`dispatchModule`
A reference to a dispatch module to be removed.

DISCUSSION

This method is not called by part editors under normal circumstances; however, your part editor may call this method to remove a custom dispatch module as a monitor for the specified event type.

SEE ALSO

The `ODEventType` type (page 862).
The `ODDispatcher::AddMonitor` method (page 112).

SetIdleFrequency

Mac OS

The `SetIdleFrequency` method sets the idle frequency for the specified part or frame.

```
void SetIdleFrequency (in ODPart part,
                       in ODFrame frame,
                       in ODIdleFrequency frequency);
```

`part`	A reference to a part that is to receive null events.
`frame`	A reference to a frame that is to receive null events, or `kODNULL` if the part as a whole is to receive idle time.
`frequency`	The idle frequency, expressed in ticks (60ths of a second).

DISCUSSION

Your part calls this method to adjust the idle frequency after a frame is registered to receive idle time.

SEE ALSO

The `ODIdleFrequency` type (page 862).
The `ODDispatcher::RegisterIdle` method (page 119).
The `ODDispatcher::UnregisterIdle` method (page 123).

SetMouseRegion

Mac OS

The `SetMouseRegion` method assigns the mouse region used in dispatching mouse-enter, mouse-within, and mouse-leave events.

```
void SetMouseRegion (in ODRgnHandle area);
```

`area` The mouse region, expressed in QuickDraw global coordinates.

DISCUSSION

Your part calls this method to set the mouse region that the dispatcher uses to report mouse-enter, mouse-within, or mouse-leave events. If you do not call this method, the mouse region defaults to a 1-by-1 pixel area.

SEE ALSO

The `ODRgnHandle` type (page 854).
The `ODDispatcher::GetMouseRegion` method (page 116).

ShouldExit

Document shell

The `ShouldExit` method returns a Boolean value that indicates whether the document shell should terminate.

```
ODBoolean ShouldExit ();
```

return value `kODTrue` if the document shell should terminate, otherwise `kODFalse`.

DISCUSSION

The document shell and container applications call this method to see whether the dispatcher recommends that their main event loop be terminated.

SEE ALSO

The `ODDispatcher::Exit` method (page 114).

UnregisterIdle

Mac OS

The `UnregisterIdle` method removes the specified part and frame from the list of parts interested in receiving idle time.

```
void UnregisterIdle (in ODPart part,
                     in ODFrame frame);
```

`part`	A reference to a part that was receiving null events.
`frame`	A reference to a frame that was receiving null events, or `kODNULL` if the part as a whole was receiving idle time.

DISCUSSION

Your part calls this method when a frame is closed or removed.

SEE ALSO

The `ODDispatcher::SetIdleFrequency` method (page 121).
The `ODDispatcher::RegisterIdle` method (page 119).

Yield

Mac OS

The `Yield` method gives processor time to other parts.

```
void Yield (in ODFrame frame);
```

`frame` A reference to a frame that yields the processor time.

DISCUSSION

Your part calls this method during a long-running operation.

ODDispatchModule

Superclasses `ODObject`

Subclasses none

An object of the `ODDispatchModule` class is used to distribute one or more event types.

Description

Part editors do not receive events directly from the operating system; rather, OpenDoc receives events, interprets them, and dispatches them, using the dispatcher, to the appropriate part editor. The dispatcher uses dispatch module objects to dispatch specific events to individual parts. The dispatcher uses at least one internal dispatch module to handle standard events—such as mouse clicks, keystrokes, and menu commands—of a particular platform. Typically, you do not need to subclass `ODDispatchModule` or even access the internal dispatch module directly. However, you can extend the OpenDoc dispatching system by installing additional dispatch modules.

The `ODDispatchModule` class is an abstract superclass that you can subclass to create a dispatch module. You can define dispatch modules to distribute new types of events or messages, such as the custom events generated by a data glove or pen, to your part editor. Each dispatch module is responsible for determining the frame or part to which it dispatches an event.

You can create a dispatch module object either from within a shell plug-in or from within one of your part's methods that are called during startup.

Your part editor can also install a dispatch module as a monitor, using the dispatcher's `AddMonitor` method (page 112). In this case, you can monitor the event stream without interfering with it. By registering a dispatch module as a monitor for a specified event type, OpenDoc notifies the dispatch module when an event of that type occurs. You might use a monitor in a debugging environment to monitor events and display a log of the events in a window. There can be one or more monitors for each event type.

For more information related to the dispatcher, see the `ODDispatcher` class description (page 108). For more information on event handling in OpenDoc, see the chapter on user events in the *OpenDoc Programmer's Guide for the Mac OS*.

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODDispatchModule`.

somInit

The `somInit` method initializes the instance variables in a System Object Model™ (SOM™) object; it is inherited from the `SOMObject` class.

If you subclass `ODDispatchModule`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in this dispatch module object. The SOM library calls this method when this dispatch module is created. You must not do anything that might fail in this method. This limits you to operations like setting pointer variables to null, setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in this dispatch module's subclass-specific initialization method; see also the `InitDispatchModule` method (page 128).

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODDispatchModule`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for this dispatch module object, including any storage related to additional instance

variables initialized in this dispatch module object. The SOM library calls this method when this dispatch module is deleted; this method must not fail.

Methods

This section presents summary descriptions of the `ODDispatchModule` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

`InitDispatchModule`	Initializes this dispatch module object.
`Dispatch`	Should dispatch the specified event to the appropriate part.

Dispatch

The `Dispatch` method should dispatch the specified event to the appropriate part.

```
ODBoolean Dispatch (inout ODEventData event,
                    inout eventinfo eventinfo);
```

`event`	A platform-specific structure representing an event. On return, the fields of the structure may have been modified. On the Mac OS platform, the structure is defined as a Mac OS event record.
`eventInfo`	A platform-specific structure that contains additional event information. On return, the relevant fields of the structure are filled in if the event was handled.
return value	`kODTrue` if the event was handled by a part, otherwise `kODFalse`.

DISCUSSION

The `eventInfo` structure contains the following information:

- a reference to an embedded frame and an embedded facet of the part (for event types `kODEvtMouseDownEmbedded`, `kODEvtMouseUpEmbedded`, or `kODEvtMouseDownBorder`)
- an `ODPoint` value describing the location of the event, expressed in frame coordinates
- a Boolean value that indicates whether an embedded part propagated an event to its containing part

OpenDoc calls this method after it has located this dispatch module in its dispatch module dictionary. This method in turn calls the part's `HandleEvent` method to give the part the opportunity to handle the specified event.

OVERRIDING

If you subclass `ODDispatchModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODEventData` type (page 860).
The `ODEventInfo` type (page 861).
The `ODDispatcher::Dispatch` method (page 113).
The `ODPart::HandleEvent` method (page 506).

InitDispatchModule

The `InitDispatchModule` method initializes this dispatch module object.

```
void InitDispatchModule (in ODSession session);
```

`session` A reference to the current session object.

DISCUSSION

This method is not called directly to initialize this dispatch module object, but is called by a subclass-specific initialization method. By convention, every subclass of `ODDispatchModule` should have a separate initialization method (for example, the `InitMyDispatchModule` method) that is called when an instance of that subclass is created. The override method may have additional parameters beyond those of the `InitDispatchModule` method. The `InitMyDispatchModule` method should call the inherited `InitDispatchModule` method at the beginning of its implementation.

If you subclass `ODDispatchModule`, your subclass-specific initialization method, rather than its `somInit` method, should handle any initialization code that can potentially fail. For example, your initialization method may attempt to allocate memory for your dispatch module.

OVERRIDING

If you subclass `ODDispatchModule`, you should not override this method.

ODDocument

Superclasses `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODDocument` class is used to represent a document and manipulate its drafts.

Description

The `ODDocument` class is implemented differently for different platforms and storage mechanisms.

The set of related classes, `ODContainer` (page 96), `ODDocument`, `ODDraft` (page 145), and `ODStorageUnit` (page 641) is called a **container suite**. Container suite classes are implemented as an integrated set for each platform and storage mechanism because they work intimately with one another at many levels. The container suite used by default on the Mac OS platform is the Bento container suite.

An OpenDoc container can contain one or more documents. Each document, in turn, can contain one or more drafts, and each draft can contain one or more storage units. Each container, document, draft, and storage unit has a unique ID.

The document shell or container application creates or accesses a document object by calling the `AcquireDocument` method (page 98) of the appropriate container. You can obtain a reference to the document's container by calling its `GetContainer` method (page 140). Each document object has a unique name within its container. The `GetName` (page 141) and `SetName` (page 143) methods retrieve and set this name.

Because each draft corresponds to a version of its document, a document can be defined as a collection of versioned drafts. The document shell or container application creates or accesses drafts by calling the document's `CreateDraft` (page 137), `AcquireDraft` (page 134), and `AcquireBaseDraft` (page 132) methods. Other `ODDocument` methods copy drafts between documents and

discard unwanted drafts. `ODDocument` is responsible for ensuring that there is only one draft object for each draft in a document.

Each draft has permissions that control access to it. Drafts are created with exclusive read/write permissions. The document shell or container application can change a draft's permissions when it calls the document's `AcquireDraft` and `AcquireBaseDraft` methods. Access to a draft is guaranteed to be exclusive only if the draft has exclusive read/write permissions.

Drafts are linearly derived in a document. The drafts of a document can be thought of as a stack; the oldest draft, called the **base draft,** is at the bottom of the stack and the most recent is at the top. A given draft is said to be above an earlier draft and below a more recent draft. The stack of drafts in the document is called the document's draft history. Although part editors can access any draft of a document, only the most recent (topmost) draft can be modified; all earlier drafts are read only.

For more information on how `ODDocument` and other container-suite classes are used, see the chapters on storage and OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Note
Parts are rarely involved with the manipulation of drafts in a document. OpenDoc and the user cooperate to create and manipulate drafts. ◆

Methods

This section presents summary descriptions of the `ODDocument` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are typically called by the document shell or container applications.

Draft Creation

`CreateDraft` [D]	Creates a new most recent draft object in this document.
`Exists`	Returns a Boolean value that indicates whether the specified draft exists in this document.

Draft Retrieval and Manipulation

`CollapseDrafts` [D] Removes the specified range of empty drafts from this document.

`AcquireBaseDraft` [D] Returns a reference to the base draft of this document with its permissions set as specified.

`AcquireDraft` [D] Returns a reference to the specified draft of this document with its permissions set as specified.

`SaveToAPrevDraft` [D] Consolidates the changes in the specified range of drafts.

`SetBaseDraftFromForeignDraft` [D] Copies a draft from another document to the base draft of this document.

`GetContainer` Returns a reference to the container object that created this document.

`GetID` Returns the unique ID of this document.

`GetName` Returns the name of this document.

Naming

`SetName` [D] Sets the name of this document.

AcquireBaseDraft

Document shell

The `AcquireBaseDraft` method returns a reference to the base draft of this document with its permissions set as specified.

```
ODDraft AcquireBaseDraft (in ODDraftPermissions perms);
```

`perms` The target permissions for the draft. Valid values for `perms` are dependent on the container suite; the Bento container suite supports two values: read only (`kODDPReadOnly`) and exclusive read/write (`kODDPExclusiveWrite`).

return value A reference to the base draft of this document.

DISCUSSION

This method returns a reference to the base draft for this document with its permissions set as specified in the `perms` parameter. If this document did not already have a base draft, this method creates, initializes, and stores the base draft.

The permissions specified by the `perms` parameter must be consistent with the ways in which other objects are currently accessing the base draft. The restrictions placed on the permissions depend on the container suite. With the Bento container suite, the permissions may be set to exclusive read/write (`kODDPExclusiveWrite`) if the base draft is the document's only draft and no other object already has access to the draft (either read only or exclusive read/write). The permissions may be set to read only (`kODDPReadOnly`) if no object has exclusive read/write access to the draft.

This method increments the reference count of the returned draft. When the caller has finished using that draft, it should call the draft's `Release` method.

EXCEPTIONS

`kODErrInvalidPermissions`
: The base draft is not accessible with the specified permissions.

SEE ALSO

The `ODDraftPermissions` type (page 872).
The `ODDocument::AcquireDraft` method (page 134).

AcquireDraft

Document shell

The `AcquireDraft` method returns a reference to the specified draft of this document with its permissions set as specified.

```
ODDraft AcquireDraft (in ODDraftPermissions perms,
                      in ODDraftID id,
                      in ODDraft draft,
                      in ODPositionCode posCode,
                      in ODBoolean release);
```

`perms`	The target permissions for the draft. Valid values for `perms` are dependent on the container suite; the Bento container suite supports two values: read only (`kODDPReadOnly`) and exclusive read/write (`kODDPExclusiveWrite`).
`id`	The ID of the desired draft, or `kODNULLID` to use the `draft` and `posCode` parameters to identify the desired draft.
`draft`	A reference to the draft object that is used with the `posCode` parameter to identify the desired draft, or `kODNULL` to use the `id` parameter to identify the desired draft.
`posCode`	The position, relative to the `draft` parameter, of the desired draft in this document's draft history, or `kODPosUndefined` to use the `id` parameter to identify the desired draft.
`release`	`kODTrue` if the draft specified by the `draft` parameter should be released after the desired draft is returned, otherwise `kODFalse`. (Used only when the `draft` and `posCode` parameters identify the desired draft.)
return value	A reference to the specified draft object.

DISCUSSION

If the `id` parameter is not null, this method uses that parameter to identify the desired draft, and the `draft`, `posCode`, and `release` parameters are ignored.

If the `id` parameter is `kODNULLID`, this method uses the `draft` and `posCode` parameters to identify the desired draft. When the `posCode` parameter is

kODPosSame, the desired draft is the draft object specified by the draft parameter, and this method changes the permissions of that draft. If the release parameter is true, the draft passed in the draft parameter is released if the desired draft can be returned.

Once the desired draft is identified, this method returns a reference to that draft object with its permissions set as specified in the perms parameter. The specified permissions must be consistent with the ways in which other objects are currently accessing the desired draft. The restrictions placed on the permissions depend on the container suite. With the Bento container suite, the permissions may be set to exclusive read/write (kODDPExclusiveWrite) if the desired draft is the document's most recent draft and no other object already has access to the draft (either read only or exclusive read/write). The permissions may be set to read only (kODDPReadOnly) if no object has exclusive read/write access to the draft.

This method increments the reference count of the returned draft. When the caller has finished using that draft, it should call the draft's Release method.

EXCEPTIONS

kODErrCannotChangePermissions
: The draft's permissions cannot be changed if the draft is retrieved with different permissions.

kODErrInvalidDraftID
: The specified draft ID is invalid.

kODErrInvalidPermissions
: The target permissions are invalid.

kODErrRefCountNotEqualOne
: Attempt to change the permissions for a draft while other objects have references to it.

kODErrUnsupportedPosCode
: The specified position code is not supported.

SEE ALSO

The ODDraftID type (page 872).
The ODDraftPermissions type (page 872).
The ODPositionCode type (page 885).
The ODContainer::AcquireDocument method (page 98).
The ODDocument::AcquireBaseDraft method (page 132).

CollapseDrafts

Document shell

The `CollapseDrafts` method removes the specified range of empty drafts from this document.

```
ODDocument CollapseDrafts (in ODDraft from,
                           in ODDraft to);
```

`from`	A reference to the first (most recent) draft in the range.
`to`	A reference to the last (oldest) draft in the range.
return value	A reference to this document with the specified drafts removed.

DISCUSSION

If the `to` parameter is `kODNULL`, it is set to the draft immediately previous to (below) the `from` draft. When successful, this method removes all the drafts between the `from` draft (inclusive) and the `to` draft (exclusive), and returns a reference to the amended document object. This method maintains the appropriate draft topology for this document. After successful execution, the `from` draft is no longer a valid draft object.

For this method to be successful, the following conditions must all be met:

- The `from` draft itself must have a reference count of 1, meaning that only the caller has a reference to that draft object.
- The `from` draft must not be this document's base draft and must be more recent than (above) the `to` draft in this document's draft history.
- All the drafts targeted for removal must be empty. A draft is empty if it contains no changes from its previous draft.
- There must be no outstanding drafts between the `from` draft (exclusive) and the `to` draft (exclusive). An outstanding draft has a reference count greater than 0, meaning that it is being used by some object.

Before calling this method, the document shell or container application can call the `SaveToAPrevDraft` method to consolidate the changes in a range of drafts.

EXCEPTIONS

`kODErrCannotCollapseDrafts`
: The `from` draft is not a more recent draft than the `to` draft.

`kODErrNonEmptyDraft`
: There are non-empty draft(s) between the `from` draft (inclusive) and the `to` draft (exclusive).

`kODErrOutstandingDraft`
: There are outstanding drafts between the `from` draft (exclusive) and the `to` draft (exclusive) or the `from` draft has a reference count greater than 1.

SEE ALSO

The `ODDocument::SaveToAPrevDraft` method (page 142).

CreateDraft

Document shell

The `CreateDraft` method creates a new most recent draft object in this document.

```
ODDraft CreateDraft (in ODDraft below,
                     in ODBoolean releaseBelow);
```

`below`
: A reference to the most recent draft for this document.

`releaseBelow`
: `kODTrue` if the `below` draft should be released, otherwise `kODFalse`.

return value
: A reference to the newly created draft object with exclusive read/write permissions.

DISCUSSION

This method creates, initializes, and returns a new draft object, which is the most recent draft of this document. (To create the base draft of the document, the document shell calls the `AcquireBaseDraft` method instead.)

If the `releaseBelow` parameter is true, this document removes its reference to the `below` draft by calling that draft's `Release` method.

This method initializes the reference count of the returned draft. When the caller has finished using that draft, it should call the draft's `Release` method.

EXCEPTIONS

`kODErrInvalidBelowDraft`
: The document has previous drafts and the `below` draft is not the most recent draft of the document.

`kODErrInvalidPermissions`
: The `below` draft has exclusive read/write permissions, and the `releaseBelow` parameter is false; this combination is illegal because only the most recent draft can have write permission.

SEE ALSO

The `ODDocument::AcquireDraft` method (page 134).
The `ODContainer::AcquireDocument` method (page 98).
The `ODRefCntObject::Release` method (page 555).
The `ODDraft` class (page 145).

Exists

The `Exists` method returns a Boolean value that indicates whether the specified draft exists in this document.

```
ODBoolean Exists (in ODDraftID id,
                  in ODDraft draft,
                  in ODPositionCode posCode);
```

`id`	The draft ID, or `kODNULLID` to use the `draft` and `posCode` parameters to identify the desired draft.
`draft`	A reference to the draft object that is used with the `posCode` parameter to identify the desired draft, or `kODNULL` to use the `id` parameter to identify the desired draft.
`posCode`	The position, relative to the `draft` parameter, of the desired draft in this document's draft history, or `kODPosUndefined` to use the `id` parameter to identify the desired draft.
return value	`kODTrue` if the desired draft exists, otherwise `kODFalse`.

DISCUSSION

If the `id` parameter is not null, this method uses that parameter to identify the desired draft and ignores the `draft` and `posCode` parameters. If the `id` parameter is `kODNULLID`, this method uses the `draft` and `posCode` parameters to identify the desired draft.

EXCEPTIONS

`kODErrInsufficientInfoInParams`
No draft was specified; the `id` parameter is `kODNULLID` and the `draft` parameter is `kODNULL`.

`kODErrUnsupportedPosCode`
The specified position code is not supported.

SEE ALSO

The `ODDraftID` type (page 872).
The `ODPositionCode` type (page 885).

GetContainer

The `GetContainer` method returns a reference to the container object that created this document.

```
ODContainer GetContainer ();
```

return value A reference to the container object that created this document.

DISCUSSION

This method does not increment the reference count of the returned container. For that reason, if you cache the returned container, you should call its `Acquire` method to increment its reference count and then call its `Release` method when you are finished using it.

SEE ALSO

The `ODContainer::AcquireDocument` method (page 98).

GetID

The `GetID` method returns the unique ID of this document object.

```
ODDocumentID GetID ();
```

return value The ID of this document object.

SEE ALSO

The `ODDocumentID` type (page 872).

GetName

The `GetName` method returns the name of this document.

```
ODDocumentName GetName ();
```

return value The name of the document, or an empty sequence if the document does not have a name.

DISCUSSION

Although parts can call this method, they are not usually concerned with document names.

If this document has a name, this method returns a copy of that name. Otherwise, the `text` field of the result `ODDocumentName` structure represents an empty sequence of characters; that is, the `text` field is an `ODByteArray` structure whose `_length` field contains 0 and whose `_buffer` field contains a null pointer.

EXCEPTIONS

`kODErrInvalidPersistentFormat`
Unable to read the name from persistent storage because it is not in a recognized format.

SEE ALSO

The `ODDocumentName` type (page 872).
The `ODDocument::SetName` method (page 143).

SaveToAPrevDraft

Document shell

The `SaveToAPrevDraft` method consolidates the changes in the specified range of drafts.

```
void SaveToAPrevDraft (in ODDraft from,
                       in ODDraft to);
```

`from`	A reference to the first (most recent) draft in the range.
`to`	A reference to the last (oldest) draft in the range.

DISCUSSION

If the `to` parameter is `kODNULL`, it is set to the draft immediately previous to the `from` draft. This method copies the content of all the drafts between the `from` draft (inclusive) and the `to` draft (exclusive) to the `to` draft. It then makes all the drafts from the `from` draft (inclusive) to the `to` draft (exclusive) empty. The document shell or container application can call the `CollapseDrafts` method to delete these empty drafts from this document.

For this method to be successful there must be no outstanding drafts between the `from` draft (exclusive) and the `to` draft (exclusive). An outstanding draft has a reference count greater than 0, meaning that it is being used by an object.

EXCEPTIONS

`kODErrDraftDoesNotExist`	Either the `from` draft does not exist, the `to` draft does not exist, or the `from` draft is not a later draft than the `to` draft.
`kODErrOutstandingDraft`	There are outstanding drafts between the `from` draft (exclusive) and the `to` draft (exclusive).

SEE ALSO

The `ODDocument::CollapseDrafts` method (page 136).
The `ODDraft::SaveToAPrevious` method (page 179).

SetBaseDraftFromForeignDraft

Document shell

The `SetBaseDraftFromForeignDraft` method copies a draft from another document to the base draft of this document.

```
void SetBaseDraftFromForeignDraft (in ODDraft draft);
```

`draft` A reference to the draft of some other document to copy to the base draft of this document.

DISCUSSION

The document shell or container application can call this method when this document is newly created to set this document's base draft to a copy of a draft of another document.

SetName

Document shell

The `SetName` method sets the name of this document.

```
void SetName (in ODDocumentName name);
```

`name` The new name for this document.

SEE ALSO

The `ODDocumentName` type (page 872).
The `ODDocument::GetName` method (page 141).

ODDraft

Superclasses `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODDraft` class represents a particular version of a document object.

Description

The `ODDraft` class is implemented differently for different platforms and storage mechanisms.

The set of related classes, `ODContainer` (page 96), `ODDocument` (page 130), `ODDraft`, and `ODStorageUnit` (page 641) is called a **container suite**. Container suite classes are implemented as an integrated set for each platform and storage mechanism because they work intimately with one another at many levels. The container suite used by default on the Mac OS platform is the Bento container suite.

An OpenDoc document can contain one or more drafts and each draft can contain one or more storage units. Each document, draft, and storage unit has a unique ID. If a storage unit is used to store data for a persistent object, the storage unit's ID is also used as the ID of the persistent object.

Because each draft corresponds to a version of its document, a document can be defined as a collection of versioned drafts. The document shell or container application creates or accesses a draft of a document by calling the `CreateDraft` (page 137), `AcquireDraft` (page 134), and `AcquireBaseDraft` (page 132) methods of that document.

Each draft has permissions that control access to it. Drafts are created with exclusive read/write permissions. The document shell or container application can change a draft's permissions when it calls the document's `AcquireDraft` and `AcquireBaseDraft` methods. Access to a draft is guaranteed to be exclusive only if the draft has exclusive read/write permissions. Many methods of `ODDraft` require a particular kind of access to the draft; for

example, the `CreateFrame` method requires write access. Before calling one of these methods, you should call the draft's `GetPermissions` method (page 170) and check that the draft's current permissions allow the required access.

Drafts are linearly derived in a document. The drafts of a document can be thought of as a stack; the oldest draft, called the base draft, is at the bottom of the stack and the most recent is at the top. Although part editors can access any draft of a document, only the most recent (topmost) draft can be modified; all earlier drafts are read only.

The `ODDraft` class has a set of `Create`*Class* and `Acquire`*Class* methods for creating and retrieving persistent objects of various classes: parts, frames, link objects, and link-source objects. For example, the `CreateFrame` method (page 162) creates a new frame object matching the specifications you provide; the `AcquireFrame` method (page 151) re-creates the `ODFrame` object stored in the specified storage unit.

Many OpenDoc methods modify the content of a draft, and therefore the draft must be notified so that it can save these changes. If this draft contains no changes since it was last saved, it is said to be **clean;** if it contains changes, it is said to be **dirty.** For example, after a successful call to the `CreateFrame` method, the draft is dirty. You can call the `SetChangedFromPrev` method (page 180) when your part's content has changed; that method marks the draft as dirty so that content changes will be saved.

You can copy or **clone** the persistent objects in a draft into a specified storage unit. All cloning must be performed within a transaction; the Bento container suite allows only one cloning transaction at any given time.

- To initiate a cloning transaction, call the draft's `BeginClone` method (page 157). This method returns a unique **draft key** that identifies the transaction.
- To copy a persistent object, call the draft's `Clone` method (page 159), passing as parameters the draft key returned by the `BeginClone` method, the ID of the object to be cloned, the ID of the destination storage unit, and the ID of the frame object specifying the scope of the cloning operation.

  The draft's `Clone` method calls the `CloneInto` method of the object being cloned. If that object has persistent references, it clones any referenced objects within the specified scope. Objects referenced by strong persistent references are strongly cloned by recursive calls to the `Clone` method;

objects referenced by weak persistent references are weakly cloned by calls to the `WeakClone` method (page 181).

- To commit the cloning transaction, call the draft's `EndClone` method (page 167). Until the `EndClone` method has successfully executed, there is no guarantee that all the objects have been copied.
- To terminate the transaction unsuccessfully, call the draft's `AbortClone` method (page 149). You should call this method if the cloning operation cannot be completed for any reason.

For more information on how `ODDraft` and other container-suite classes are used, see the chapters on storage and OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODDraft` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are called only by the document shell or container applications.

Draft Characteristics

`GetDocument` [D] Returns a reference to the document object that created this draft.

`AcquireDraftProperties` Returns a reference to the storage unit in which this draft stores its properties and data that are global to this draft.

`GetID` [D] Returns the draft ID of this draft.

`IsValidID` Returns a Boolean value that indicates whether the specified object ID is valid.

`GetPermissions` Returns this draft's current permissions.

`GetPersistentObjectID` Returns the scripting ID of the specified part or frame.

`AcquirePersistentObject` Returns a reference to the part or frame with the specified scripting ID.

Object Factory Methods

`CreateFrame`	Creates a new frame in this draft.
`AcquireFrame`	Returns a reference to the frame whose data is stored in the specified storage unit.
`AcquireLink`	Returns a reference to the link object whose data is stored in the specified storage unit or that can be constructed from the given link specification.
`CreateLinkSource`	Creates a new link-source object in this draft.
`AcquireLinkSource`	Returns a reference to the link-source object whose data is stored in the specified storage unit.
`CreateLinkSpec`	Creates a link-specification object for the specified part.
`CreatePart`	Creates a new part in this draft.
`AcquirePart`	Returns a reference to the part whose data is stored in the specified storage unit.
`CreateStorageUnit`	Creates a new storage unit in this draft.
`AcquireStorageUnit`	Returns a reference to the storage unit with the specified ID.

Cloning

`BeginClone`	Initiates a cloning transaction to transfer data from this draft to the specified destination draft.
`Clone`	Clones the specified persistent object or storage unit.
`WeakClone`	Ensures that, if the specified object is cloned, weak persistent references to it are maintained.
`EndClone`	Commits the specified cloning transaction.
`AbortClone`	Aborts the specified cloning transaction.

Modifying Drafts

`Externalize` [D]	Writes to storage all persistent objects and storage units of this draft object.
`RemoveChanges` [D]	Removes all changes made in this draft.
`RemoveFromDocument` [D]	Removes this draft from its document and destroys this draft.
`ReleasePart`	Releases the specified part of this draft.

`SaveToAPrevious` [D]	Copies the content of this draft to the specified previous draft of the same document.
`ChangedFromPrev` [D]	Returns a Boolean value that indicates whether this draft contains any changes from the previous draft.
`SetChangedFromPrev`	Marks this draft as dirty.

Internal Methods

`RemoveFrame`	Removes the specified frame from this draft.
`RemoveLink`	Removes the specified link object from this draft.
`RemoveLinkSource`	Removes the specified link-source object from this draft.
`RemovePart`	Removes the specified part from this draft.
`RemoveStorageUnit`	Removes the specified storage unit from this draft.

AbortClone

The `AbortClone` method aborts the specified cloning transaction.

```
void AbortClone (in ODDraftKey key);
```

`key` The draft key of the cloning transaction to be aborted.

DISCUSSION

You call this method if a cloning transaction cannot be completed for any reason. The `key` parameter must be set to the draft key value returned by the call to the `BeginClone` method that started the cloning transaction.

EXCEPTIONS

`kODErrInvalidDraftKey`
The specified draft key is not the draft key for the current cloning transaction.

SEE ALSO

The `ODDraftKey` type (page 872).
The `ODDraft::BeginClone` method (page 157).
The `ODDraft::Clone` method (page 159).
The `ODDraft::EndClone` method (page 167).

AcquireDraftProperties

The `AcquireDraftProperties` method returns a reference to the storage unit in which this draft stores its properties and data that are global to this draft.

```
ODStorageUnit AcquireDraftProperties ();
```

return value A reference to this draft's draft-properties storage unit.

DISCUSSION

The draft stores information about itself in its draft-properties storage unit. In addition, your part may create properties in this storage unit to store information related to this draft.

This method increments the reference count of the returned storage unit. When you have finished using that storage unit, you should call its `Release` method.

EXCEPTIONS

`kODErrNoDraftProperties`
A draft-properties storage unit cannot be created.

AcquireFrame

The `AcquireFrame` method returns a reference to the frame whose data is stored in the specified storage unit.

```
ODFrame AcquireFrame (in ODStorageUnitID id);
```

`id`	The storage-unit ID for the target frame.
return value	A reference to the specified frame object.

DISCUSSION

This method increments the reference count of the returned frame. When you have finished using that frame, you should call its `Release` method.

EXCEPTIONS

`kODErrCannotAcquireFrame`
Cannot access the target frame object.

`kODErrIllegalNullIDInput`
The `id` parameter is null.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODDraft::CreateFrame` method (page 162).
The `ODFrame` class (page 288).

AcquireLink

The `AcquireLink` method returns a reference to the link object whose data is stored in the specified storage unit or that can be constructed from the given link specification.

```
ODLink AcquireLink (in ODStorageUnitID id,
                    in ODLinkSpec linkSpec);
```

`id`	The ID of the storage unit providing the link data, or `kODNULLID` if the link is to be constructed.
`linkSpec`	A reference to a link-specification object from which the link is to be constructed, or `kODNULL` is the `id` parameter is not null.
return value	A reference to the specified link object.

DISCUSSION

If your part is the destination of a link, you call this method to create a link object; you also call this method in your part's `InitPartFromStorage` method to re-create the link object from its stored data.

If the `id` parameter is not null, the `linkSpec` parameter is ignored. In that case, the `id` parameter must be the ID of the link object's storage unit, and this method re-creates the link object from the content of the specified storage unit.

If the `id` parameter is null, the `linkSpec` parameter must be a link-specification object returned by a previous call to the `CreateLinkSpec` method. In that case, this method uses the information in the link-specification object to construct the link object.

This method increments the reference count of the returned link object. When you have finished using that link object, you should call its `Release` method.

EXCEPTIONS

`kODErrCannotAcquireLink`
: Cannot re-create the requested link object from the specified storage unit or link-specification object.

`kODErrInsufficientInfoInParams`
: No link object was specified; the `id` parameter is `kODNULLID` and the `linkSpec` parameter is `kODNULL`.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODDraft::CreateLinkSpec` method (page 164).
The `ODPart::InitPartFromStorage` method (page 510).
The `ODLink` class (page 339).
The `ODLinkSpec` class (page 379).

AcquireLinkSource

The `AcquireLinkSource` method returns a reference to the link-source object whose data is stored in the specified storage unit.

```
ODLinkSource AcquireLinkSource (in ODStorageUnitID id);
```

`id`	The ID of the storage unit for the target link.
return value	A reference to the specified link-source object.

DISCUSSION

If your part is the source of a link, you can call this method in your part's `InitPartFromStorage` method to re-create the link-source object from its stored data.

This method increments the reference count of the returned link-source object. When you have finished using that link-source object, you should call its `Release` method.

EXCEPTIONS

`kODErrCannotAcquireLink`
: The companion link object does not exist and cannot be created.

`kODErrIllegalNullIDInput`
: The `id` parameter is null.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODDraft::CreateLinkSource` method (page 163).
The `ODPart::InitPartFromStorage` method (page 510).
The `ODLinkSource` class (page 361).

AcquirePart

The `AcquirePart` method returns a reference to the part whose data is stored in the specified storage unit.

```
ODPart AcquirePart (in ODStorageUnitID id);
```

`id`	The ID of the target part.
return value	A reference to the specified part object.

DISCUSSION

You call this method to re-create a part object from its stored data.

This method increments the reference count of the returned part. When you have finished using that part, you should call its `Release` method.

EXCEPTIONS

`kODErrIllegalNullIDInput`
: The `id` parameter is null.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODDraft::CreatePart` method (page 166).
The `ODPart` class (page 445).

AcquirePersistentObject

The `AcquirePersistentObject` method returns a reference to the part or frame with the specified scripting ID.

```
ODPersistentObject AcquirePersistentObject (
                        in ODPersistentObjectID objectID,
                        out ODObjectType objectType);
```

`objectID` The scripting ID of the desired object.

`objectType`
The type of the object returned, either part (`kODPartObject`) or frame (`kODFrameObject`).

return value A reference to the part or frame with the specified scripting ID.

DISCUSSION

You call this method to obtain a reference to a part or frame, given the ID that identifies it for scripting purposes. This method should be used only for scripting.

This method increments the reference count of the returned persistent object. When you have finished using that persistent object, you should call its `Release` method.

EXCEPTIONS

`kODErrIllegalNullIDInput`
The `objectID` parameter is null.

`kODErrInvalidPersistentObjectID`
This draft has no persistent object with the

specified ID, or the object is of a type that cannot be acquired.

SEE ALSO

The `ODObjectType` type (page 873).
The `ODPersistentObjectID` type (page 870).
The `ODDraft::GetPersistentObjectID` method (page 171).

AcquireStorageUnit

The `AcquireStorageUnit` method returns a reference to the storage unit with the specified ID.

```
ODStorageUnit AcquireStorageUnit (in ODStorageUnitID id);
```

`id`	The ID of the target storage unit.
return value	A reference to the specified storage-unit object.

DISCUSSION

You call this method to re-create a storage-unit object from its stored data. The `id` parameter must be the ID of a storage unit in this draft.

This method increments the reference count of the returned storage unit. When you have finished using that storage unit, you should call its `Release` method.

EXCEPTIONS

`kODErrIllegalNullIDInput`
The `id` parameter is null.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODDraft::CreateStorageUnit` method (page 167).
The `ODStorageUnit` class (page 641).

BeginClone

The `BeginClone` method initiates a cloning transaction to transfer data from this draft to the specified destination draft.

```
ODDraftKey BeginClone (in ODDraft destDraft,
                       in ODFrame destFrame,
                       in ODCloneKind kind);
```

`destDraft`	A reference to the destination draft to which objects are to be cloned.
`destFrame`	A reference to the destination frame, or `kODNULL` if the clone operation is not a paste or a drop.
`kind`	The kind of clone operation being performed.
return value	The draft key for this cloning transaction.

DISCUSSION

The `BeginClone` method sets up a cloning transaction of the specified kind and returns a unique draft key that identifies the transaction. Other methods involved in cloning require you to supply this draft key as a parameter. The Bento container suite allows only one cloning transaction at any given time.

The `destFrame` parameter allows OpenDoc to detect an attempt to move a part into one of its embedded frames. For pasting operations, the `destFrame` parameter should be set to the part's active frame. For dropping operations, this parameter should be set to the drop target frame. In all other cases, this parameter should be set to null.

The `kind` parameter indicates the nature of the cloning operation required, such as copy object from source to clipboard (`kODCloneCopy`), cut object from source to clipboard (`kODCloneCut`), and so on.

After calling this method, you call this draft object's `Clone` method to clone a particular persistent object. To commit a successful cloning transaction, call this draft object's `EndClone` method; before the `EndClone` method has successfully executed, there is no guarantee that all the objects have been copied. If the cloning operation cannot be completed for any reason, you must terminate the transaction by calling this draft object's `AbortClone` method.

EXCEPTIONS

`kODErrCloningInProgress`
: Another cloning process is in progress.

`kODErrInconsistentCloneKind`
: The specified clone kind is used inconsistently. For example, a paste or drop clone kind can occur only following a copy or cut operation.

`kODErrInvalidCloneKind`
: The specified clone kind is not valid.

`kODErrInvalidDestinationDraft`
: The specified destination draft is not valid for the specified clone kind.

SEE ALSO

The `ODCloneKind` type (page 887).
The `ODDraftKey` type (page 872).
The `ODDraft::AbortClone` method (page 149).
The `ODDraft::Clone` method (page 159).
The `ODDraft::EndClone` method (page 167).

ChangedFromPrev

Document shell

The `ChangedFromPrev` method returns a Boolean value that indicates whether this draft contains any changes from the previous draft.

```
ODBoolean ChangedFromPrev ();
```

return value `kODTrue` if this draft contains any changes from the previous draft, otherwise `kODFalse`.

DISCUSSION

The document shell or container application calls this method to see whether this draft contains changes from the previous draft; if so, the draft must be notified so that it can save these changes. If this draft contains no changes since it was last saved, it is said to be clean and this method returns false; if it contains changes, it is said to be dirty and this method returns true.

You can call the `SetChangedFromPrev` method when your part's content has changed; this marks the draft as dirty so that content changes will be saved.

Note that a draft cannot be marked clean after it has been marked dirty. However, you can call the `RemoveChanges` method to remove all the changes from this draft.

SEE ALSO

The `ODDraft::RemoveChanges` method (page 173).
The `ODDraft::SetChangedFromPrev` method (page 180).

Clone

The `Clone` method clones the specified persistent object or storage unit.

```
ODID Clone (in ODDraftKey key,
            in ODID fromObjectID,
            in ODID toObjectID,
            in ODID scope);
```

`key` The draft key of the current cloning transaction.

`fromObjectID` The ID of the persistent object or storage unit to be cloned.

`toObjectID` The ID of the destination persistent object or storage unit, or `kODNULLID` to create a new storage unit in the destination draft.

`scope`	The ID of the frame that defines the scope of this cloning operation.
return value	The ID of the destination persistent object or storage unit. (The ID of a persistent object is the same as the ID of its storage unit.)

DISCUSSION

The following summary describes how to use the `Clone` method.

- You start a cloning transaction by calling this draft object's `BeginClone` method, which returns the draft key identifying the cloning transaction.
- Next, call this draft object's `Clone` method, passing as parameters the draft key returned by the `BeginClone` method, the ID of the object to be cloned, the ID of the destination storage unit, and the frame object specifying the scope.
- Finally, call this draft object's `EndClone` method, passing as a parameter the draft key returned by the `BeginClone` method. The `EndClone` method commits the cloning transaction and performs the data transfer.

The `key` parameter is the draft key of the current cloning transaction, which was returned by a call to this draft object's `BeginClone` method and passed to the calling object's `CloneInto` method.

The `fromObjectID` parameter identifies the object to be cloned. If a persistent object exists with the specified ID, that persistent object is cloned; otherwise, the storage unit with the specified ID is cloned.

The `toObjectID` parameter specifies the ID of the destination storage unit. If the `toObjectID` parameter is null, a new destination storage unit is created in the destination draft.

If the object being cloned has persistent references to other objects, the `scope` parameter determines which of the referenced objects are within the scope of this cloning operation. Typically, the `scope` parameter is the ID of a frame and only those objects embedded in that frame are within scope. In the rare case in which the `scope` parameter is `kODIDAll`, all referenced objects are within scope.

This method passes its `key` parameter, the destination storage unit, and its `scope` parameter to the `CloneInto` method of the persistent object or storage unit being cloned. If that persistent object or storage unit has persistent

references, its `CloneInto` method clones any persistently referenced objects that are within the scope of this cloning operation. Objects referenced by strong persistent references are strongly cloned by recursive calls to the `Clone` method; objects referenced by weak persistent references are weakly cloned by calls to the `WeakClone` method.

You must not use the returned ID until the end of the current cloning transaction. Furthermore, before you try to access the persistent object or storage unit with the corresponding ID, you must call the `IsValidID` method to verify that the ID is still valid.

EXCEPTIONS

`kODErrInvalidDraftKey`	The specified draft key is not the draft key for the current cloning transaction.
`kODErrInvalidID`	The `toObjectID` parameter did not specify a valid destination object or storage unit.

SEE ALSO

The `ODDraftKey` type (page 872).
The `ODID` type (page 869).
The `ODDraft::AbortClone` method (page 149).
The `ODDraft::BeginClone` method (page 157).
The `ODDraft::EndClone` method (page 167).
The `ODDraft::IsValidID` method (page 172).
The `ODDraft::WeakClone` method (page 181).
The `ODPersistentObject::CloneInto` method (page 536).
The `ODStorageUnit::CloneInto` method (page 650).

CreateFrame

The `CreateFrame` method creates a new frame in this draft.

```
ODFrame CreateFrame (in ODObjectType frameType,
                     in ODFrame containingFrame,
                     in ODShape frameShape,
                     in ODCanvas biasCanvas,
                     in ODPart part,
                     in ODTypeToken viewType,
                     in ODTypeToken presentation,
                     in ODBoolean isSubframe,
                     in ODBoolean isOverlaid);
```

`frameType` The type of the frame to be created. The frame type must be either a regular frame (`kODFrameObject`) or a nonpersistent frame (`kODNonPersistentFrameObject`).

`containingFrame`
A reference to the containing frame of the frame being created.

`frameShape` A reference to the frame shape to be created.

`biasCanvas` A reference to the canvas to whose coordinate space the new frame is biased, or `kODNULL` to use the standard platform coordinate bias.

`part` A reference to the part to be displayed in the new frame.

`viewType` A tokenized string representing the initial view type for the new frame.

`presentation`
The initial presentation for the new frame.

`isSubframe` `kODTrue` if the new frame is a subframe, otherwise `kODFalse`.

`isOverlaid` `kODTrue` if the new frame should be an overlaid frame, otherwise `kODFalse`.

return value A reference to the newly created frame object.

DISCUSSION

This method constructs and returns a frame object in this draft; the new frame has the characteristics specified by parameters. You can create a regular frame (`kODFrameObject`) only if this draft's current permissions provide write access.

The `viewType` parameter must be the tokenized form of one of the view-type constants (`kODViewAsFrame`, `kODViewAsLargeIcon`, `kODViewAsSmallIcon`, or `kODViewAsThumbnail`). You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

This method initializes the reference count of the returned frame. When you have finished using that frame, you should call its `Release` method.

If this method executes successfully, it marks this draft as dirty.

EXCEPTIONS

`kODErrCannotCreateFrame`
: Cannot create the frame object.

`kODErrInvalidObjectType`
: The specified frame type is not `kODFrameObject` or `kODNonPersistentFrameObject`.

SEE ALSO

The `ODObjectType` type (page 873).
The `ODTypeToken` type (page 847).
The `ODDraft::AcquireFrame` method (page 151).
The `ODSession::Tokenize` method (page 598).
The `ODFrame` class (page 288).

CreateLinkSource

The `CreateLinkSource` method creates a new link-source object in this draft.

```
ODLinkSource CreateLinkSource (in ODPart part);
```

`part`	A reference to the part containing the source content of the link.
return value	A reference to the newly created link-source object.

DISCUSSION

You typically call this method from your part's `CreateLink` method when creating the source of a link. You can call this method only if this draft's current permissions provide write access.

Because all link-source objects have a unique link object companion, this method fails if the companion object cannot be created.

This method initializes the reference count of the returned link-source object. When you have finished using that link-source object, you should call its `Release` method.

If this method executes successfully, it marks this draft as dirty.

EXCEPTIONS

`kODErrCannotCreateLink`
: Cannot create the link-source object or the companion link object.

SEE ALSO

The `ODDraft::AcquireLinkSource` method (page 153).
The `ODPart::CreateLink` method (page 476).
The `ODLinkSource` class (page 361).

CreateLinkSpec

The `CreateLinkSpec` method creates a link-specification object for the specified part.

```
ODLinkSpec CreateLinkSpec (in ODPart part,
                           in ODByteArray data);
```

`part`	A reference to the source part creating the link specification, or `kODNULL` to create an empty link specification.
`data`	A byte array whose buffer contains arbitrary data to be placed in the link specification, or `kODNULL` to create an empty link specification.
return value	A reference to the newly created link-specification object.

DISCUSSION

For a link to be created, this method must be called by both the source part and the destination part of a data transfer operation.

- A source part that supports linking calls this method to create a link-specification object. The `part` parameter is a reference to the source part. The buffer of the `data` parameter can contain any data that the source part's `CreateLink` method may need for creating the specified link.

  In addition to writing the source content to the clipboard or drag-and-drop object, the source part calls the `WriteLinkSpec` method of the returned link-specification object to write link-specification data as a signal that the source can create a link.

  After calling this method, the source part is responsible for deallocating the `data` parameter and its buffer.

- The user who pastes or drops the source data to a destination part can request a link by means of the Paste As dialog box. When this happens, the destination part calls this method to create an empty link-specification object. The `part` and `data` parameters are both null. The destination part then calls the `ReadLinkSpec` method of the returned link-specification object to read the link-specification data. The destination part creates the link by passing the link-specification object to its draft's `AcquireLink` method.

The calling part should delete the link-specification object when it has finished using it.

If this method executes successfully, it marks this draft as dirty.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODDraft::AcquireLink` method (page 152).

The `ODLinkSpec::ReadLinkSpec` method (page 381).
The `ODLinkSpec::WriteLinkSpec` method (page 382).
The `ODPart::CreateLink` method (page 476).
The `ODLink` class (page 339).
The `ODLinkSpec` class (page 379).

CreatePart

The `CreatePart` method creates a new part in this draft.

```
ODPart CreatePart (in ODType partType,
                   in ODEditor optionalEditor);
```

`partType` The part kind for the new part, or `kODNULL` if you will set the part kind later.

`optionalEditor`
The part editor to be used if the normal binding fails, or `kODNoEditor` if you choose not to specify an editor.

return value A reference to the newly created part object.

DISCUSSION

You can call this method only if this draft's current permissions provide write access.

This method calls the new part's `InitPart` method to initialize the part object.

This method initializes the reference count of the returned part. When you have finished using that part, you should call its `Release` method.

If this method executes successfully, it marks this draft as dirty.

EXCEPTIONS

`kODErrCannotCreatePart`
Cannot create the specified part object.

SEE ALSO

The `ODType` type (page 846).
The `ODEditor` type (page 899).
The `ODDraft::AcquirePart` method (page 154).
The `ODPart::InitPart` method (page 509).
The `ODPart` class (page 445).

CreateStorageUnit

The `CreateStorageUnit` method creates a new storage unit in this draft.

```
ODStorageUnit CreateStorageUnit ();
```

return value A reference to the newly created storage-unit object.

DISCUSSION

You can call this method only if this draft's current permissions provide write access.

This method initializes the reference count of the returned storage unit. When you have finished using that storage unit, you should call its `Release` method.

If this method executes successfully, it marks this draft as dirty.

SEE ALSO

The `ODDraft::AcquireStorageUnit` method (page 156).
The `ODStorageUnit` class (page 641).

EndClone

The `EndClone` method commits the specified cloning transaction.

```
void EndClone (in ODDraftKey key);
```

`key`	The draft key for the cloning transaction to be committed.

DISCUSSION

You must call this method to end a successful cloning transaction. The `key` parameter must be set to the draft key returned by the call to the `BeginClone` method that started the cloning transaction.

If the cloning transaction cannot be completed for any reason, you should call the `AbortClone` method instead of this method.

After this method executes successfully, the destination draft of this cloning transaction contains copies of all the persistent objects and storage units whose `CloneInto` methods were called by this draft's `Clone` method during the transaction.

EXCEPTIONS

`kODErrInvalidDraftKey`	The specified draft key is not the draft key for the current cloning transaction.
`kODErrMoveIntoSelf`	This clone transaction attempted to move a part into one of its embedded frames or embed one of the part's display frames into another of its display frames.

SEE ALSO

The `ODDraftKey` type (page 872).
The `ODDraft::AbortClone` method (page 149).
The `ODDraft::BeginClone` method (page 157).
The `ODDraft::Clone` method (page 159).

Externalize

Document shell

The `Externalize` method writes to storage all persistent objects and storage units of this draft object.

```
ODDraft Externalize ();
```

return value A reference to this draft object.

DISCUSSION

This method calls the `Externalize` methods on all the persistent objects and storage-unit objects created through this draft and then writes to persistent storage any internal structures of this draft object.

SEE ALSO

The discussion of overriding the `Externalize` method when you subclass `ODPart` (page 455).
The `ODPersistentObject::Externalize` method (page 537).
The `ODStorageUnit::Externalize` method (page 660).

GetDocument

Document shell

The `GetDocument` method returns a reference to the document object that created this draft.

```
ODDocument GetDocument ();
```

return value A reference to the document object that created this draft.

DISCUSSION

This method does not increment the reference count of the returned document; the caller should not call that document's `Release` method.

SEE ALSO

The `ODDocument` class (page 130).

GetID

Document shell

The `GetID` method returns the draft ID of this draft.

```
ODDraftID GetID ();
```

return value The ID of this draft.

SEE ALSO

The `ODDraftID` type (page 872).

GetPermissions

The `GetPermissions` method returns this draft's current permissions.

```
ODDraftPermissions GetPermissions ();
```

return value The permissions on this draft.

DISCUSSION

The returned value specifies the kind of access that is currently allowed to this draft:

- Exclusive read/write access (`kODDPExclusiveWrite`).
- Shared read/write access (`kODDPSharedWrite`).
- Read-only access (`kODDPReadOnly`).
- Navigation-only access (`kODDPTransient`).
- No access (`kODDPNone`).

You can make changes to this draft only if its current permissions allow write access (exclusive or shared). Only the most recent draft of a document can allow write access.

The Bento container suite supports only the `kODDPReadOnly` and `kODDPExclusiveWrite` draft permissions.

GetPersistentObjectID

The `GetPersistentObjectID` method returns the scripting ID of the specified part or frame.

```
ODPersistentObjectID GetPersistentObjectID (
                        in ODPersistentObject object,
                        in ODObjectType objectType);
```

`object`	A reference to the persistent object.
`objectType`	The type of the persistent object. The object type must be either a part (`kODPartObject`) or a frame (`kODFrameObject`).
return value	The ID, to be used for scripting purposes, of the specified part or frame.

DISCUSSION

You can call this method to obtain the ID that identifies the specified part or frame for scripting purposes.

EXCEPTIONS

`kODErrInvalidPersistentObject`
This specified persistent object is not valid.

SEE ALSO

The `ODObjectType` type (page 873).
The `ODPersistentObjectID` type (page 870).
The `ODDraft::AcquirePersistentObject` method (page 155).

IsValidID

The `IsValidID` method returns a Boolean value that indicates whether the specified object ID is valid.

```
ODBoolean IsValidID (in ODID id);
```

`id`	The object ID.
return value	`kODTrue` if the specified object ID is valid, otherwise `kODFalse`.

DISCUSSION

You should call this method after the end of a cloning transaction and before using an ID returned by the `Clone` or `WeakClone` method during that transaction.

The `id` parameter is an ID returned by the `Clone` or `WeakClone` method during the recent cloning transaction. If the corresponding object was not strongly cloned during the transaction, the ID is now invalid and should not be used.

SEE ALSO

The `ODID` type (page 869).
The `ODDraft::Clone` method (page 159).
The `ODDraft::WeakClone` method (page 181).

ReleasePart

The `ReleasePart` method releases the specified part of this draft.

```
void ReleasePart (in ODPart part);
```

`part` A reference to the part to be released.

DISCUSSION

You should call this method only from your part's `Release` method when the part's reference count is decremented to 0.

RemoveChanges

Document shell

The `RemoveChanges` method removes all changes made in this draft.

```
ODDraft RemoveChanges ();
```

return value A reference to this draft object with its changes removed.

DISCUSSION

This method can be called only if this draft's current permissions provide write access.

After this method executes successfully, this draft is empty, and a subsequent call to the `ChangedFromPrev` method returns false.

RemoveFrame

The `RemoveFrame` method removes the specified frame from this draft.

```
void RemoveFrame (in ODFrame frame);
```

`frame` A reference to the frame object to be removed.

DISCUSSION

OpenDoc calls this method internally; parts, the document shell, and container applications do not call this method.

When this method is called, the reference count of the specified frame must be 1, and this draft's current permissions must provide write access.

If this method executes successfully, it marks this draft as dirty. The specified frame is no longer a valid object, and this draft no longer contains a frame object with the corresponding ID.

EXCEPTIONS

`kODErrRefCountGreaterThanZero`
After the specified frame was released, its reference count was greater than 0.

`kODErrRefCountNotEqualOne`
The reference count of the specified frame's storage unit is not 1.

SEE ALSO

The `ODDraft::CreateFrame` method (page 162).
The `ODDraft::AcquireFrame` method (page 151).
The `ODFrame::Remove` method (page 322).
The `ODFrame` class (page 288).

RemoveFromDocument

Document shell

The `RemoveFromDocument` method removes this draft from its document and destroys this draft.

```
void RemoveFromDocument ();
```

DISCUSSION

This method can be called only if this draft is empty, is not the base draft of its document, and has a reference count of 1.

After this method executes successfully, this draft is no longer a valid draft object.

RemoveLink

The `RemoveLink` method removes the specified link object from this draft.

```
void RemoveLink (in ODLink link);
```

`link` A reference to the link object to be removed.

DISCUSSION

OpenDoc calls this method internally; parts, the document shell, and container applications do not call this method.

When this method is called, the reference count of the specified link object must be 1, and this draft's current permissions must provide write access.

If this method executes successfully, it marks this draft as dirty. The specified link object is no longer a valid object, and this draft no longer contains a link object with the corresponding ID.

EXCEPTIONS

`kODErrRefCountGreaterThanZero`
After the specified link object was released, its reference count was greater than 0.

`kODErrRefCountNotEqualOne`
The reference count of the specified link object's storage unit is not 1.

SEE ALSO

The `ODDraft::AcquireLink` method (page 152).
The `ODLink` class (page 339).

RemoveLinkSource

The `RemoveLinkSource` method removes the specified link-source object from this draft.

```
void RemoveLinkSource (in ODLinkSource link);
```

`link` A reference to the link-source object to be removed.

DISCUSSION

OpenDoc calls this method internally; parts, the document shell, and container applications do not call this method.

When this method is called, the reference count of the specified link-source object must be 1, and this draft's current permissions must provide write access.

If this method executes successfully, it marks this draft as dirty. The specified link-source object is no longer a valid object, and this draft no longer contains a link-source object with the corresponding ID.

EXCEPTIONS

`kODErrRefCountGreaterThanZero`
After the specified link-source object was released, its reference count was greater than 0.

`kODErrRefCountNotEqualOne`
The reference count of the specified link-source object's storage unit is not 1.

SEE ALSO

The `ODDraft::AcquireLinkSource` method (page 153).
The `ODLinkSource` class (page 361).

RemovePart

The `RemovePart` method removes the specified part from this draft.

```
void RemovePart (in ODPart part);
```

`part` A reference to the part object to be removed.

DISCUSSION

OpenDoc calls this method internally; parts, the document shell, and container applications do not call this method.

When this method is called, the reference count of the specified part must be 1, and this draft's current permissions must provide write access.

If this method executes successfully, it marks this draft as dirty. The specified part is no longer a valid object, and this draft no longer contains a part object with the corresponding ID.

EXCEPTIONS

`kODErrRefCountGreaterThanZero`
After the specified part was released, its reference count was greater than 0.

`kODErrRefCountNotEqualOne`
The reference count of the specified part's storage unit is not 1.

SEE ALSO

The `ODDraft::AcquirePart` method (page 154).
The `ODDraft::CreatePart` method (page 166).
The `ODPart` class (page 445).

RemoveStorageUnit

The `RemoveStorageUnit` method removes the specified storage unit from this draft.

```
void RemoveStorageUnit (in ODStorageUnit storageUnit);
```

`storageUnit`
A reference to the storage-unit object to be removed.

DISCUSSION

OpenDoc calls this method internally; parts, the document shell, and container applications do not call this method.

When this method is called, the reference count of the specified storage unit must be 1, and this draft's current permissions must provide write access.

If this method executes successfully, it marks this draft as dirty. The specified storage unit is no longer a valid object, and this draft no longer contains a storage-unit object with the corresponding ID.

EXCEPTIONS

`kODErrIllegalOperationOnSU`
: The specified storage unit cannot be removed. For example, this draft cannot remove the storage unit where it stores its draft properties.

`kODErrInvalidStorageUnit`
: The specified storage unit is not a valid storage unit.

SEE ALSO

The `ODDraft::AcquireStorageUnit` method (page 156).
The `ODDraft::CreateStorageUnit` method (page 167).
The `ODStorageUnit::Remove` method (page 682).
The `ODStorageUnit` class (page 641).

SaveToAPrevious

Document shell

The `SaveToAPrevious` method copies the content of this draft to the specified previous draft of the same document.

```
ODDraft SaveToAPrevious (in ODDraft to);
```

`to` A reference to the destination draft object, or `kODNULL` for the draft immediately previous to (below) this draft.

return value A reference to this draft object.

DISCUSSION

If the `to` parameter is null, it is set to the draft immediately previous to (below) this draft. This method copies the content of all the drafts between this draft (inclusive) and the `to` draft (exclusive) to the `to` draft. It then makes all the drafts from this draft (inclusive) to the `to` draft (exclusive) empty. The

document shell can call the `CollapseDrafts` method of this draft's document object to delete these empty drafts from the document.

For this method to be successful this draft must have a reference count of 1, meaning that only the caller has a reference to this draft object. In addition, there must be no outstanding drafts between this draft (exclusive) and the `to` draft (exclusive). An outstanding draft has a reference count greater than 0, meaning that it is being used by some object.

Calling this method is equivalent to calling the `SaveToAPrevDraft` method of this draft's document, passing this draft as the `from` parameter, and passing the same `to` parameter.

EXCEPTIONS

`kODErrOutstandingDraft`
: There are outstanding drafts between this draft (exclusive) and the `to` draft (exclusive) or this draft has a reference count greater than 1.

SEE ALSO

The `ODDocument::CollapseDrafts` method (page 136).
The `ODDocument::SaveToAPrevDraft` method (page 142).

SetChangedFromPrev

The `SetChangedFromPrev` method marks this draft as dirty.

```
void SetChangedFromPrev ();
```

DISCUSSION

A draft is said to be dirty if it contains changes since it was last saved. This method lets you mark a draft as dirty when your part's content changes, so that the content changes will be saved.

This method can be called only if this draft's current permissions provide write access.

Note that a draft cannot be marked clean after it has been marked dirty. However, you can call the `RemoveChanges` method to remove all the changes from this draft.

SEE ALSO

The `ODDraft::ChangedFromPrev` method (page 158).
The `ODDraft::RemoveChanges` method (page 173).

WeakClone

The `WeakClone` method ensures that, if the specified object is cloned, weak persistent references to it are maintained.

```
ODID WeakClone (in ODDraftKey key,
                in ODID objectID,
                in ODID toObjectID,
                in ODID scope);
```

`key`	The draft key of the current cloning transaction.
`objectID`	The ID of the persistent object or storage unit to be weakly cloned.
`toObjectID`	The ID of the destination persistent object or storage unit, or `kODNULLID` to create a new storage unit in this draft.
`scope`	The ID of the frame that defines the scope of this cloning operation.
return value	The ID of the duplicated persistent object or storage unit.

DISCUSSION

This method is called by the `CloneInto` method of persistent objects. You can call this method from your part's `CloneInto` method if your part has weak persistent references to other objects.

The `key` parameter is the draft key of the current cloning transaction, which was returned by a call to this draft object's `BeginClone` method and passed to the calling object's `CloneInto` method.

The `objectID` parameter is the ID of an object to be weakly cloned because the calling object has a weak persistent reference to it. The `WeakClone` method does not guarantee that the specified object will be copied, but if the object is copied because of an existing strong persistent reference to it, the calling object's weak persistent references will be maintained across the cloning transaction.

The `toObjectID` parameter specifies the ID of the destination storage unit. If the `toObjectID` parameter is null, a new destination storage unit is created in this destination draft, if necessary.

If the object being weakly cloned has persistent references to other objects, the `scope` parameter determines which of the referenced objects are within the scope of this cloning operation. Usually the `scope` parameter is the ID of a frame, and only those objects embedded in that frame are within scope. In the rare case in which the `scope` parameter is `kODIDAll`, all referenced objects are within scope.

You must not use the returned ID until the end of the current cloning transaction. Furthermore, before you try to access the persistent object or storage unit with the corresponding ID, you must call the `IsValidID` method to verify that the ID is still valid.

EXCEPTIONS

`kODErrInvalidID`	The `toObjectID` parameter did not specify a valid destination object or storage unit.

SEE ALSO

The `ODDraftKey` type (page 872).
The `ODID` type (page 869).
The `ODDraft::AbortClone` method (page 149).
The `ODDraft::BeginClone` method (page 157).
The `ODDraft::Clone` method (page 159).
The `ODDraft::EndClone` method (page 167).
The `ODDraft::IsValidID` method (page 172).

The `ODPersistentObject::CloneInto` method (page 536).
The `ODStorageUnit::CloneInto` method (page 650).

ODDragAndDrop

Superclasses `ODObject`

Subclasses none

An object of the `ODDragAndDrop` class provides the mechanism for dragging and dropping objects within a part, between parts, between documents, and between an OpenDoc document and a non-OpenDoc application.

Description

The drag-and-drop facility of OpenDoc depends on system services provided by the platform. They include a service to allow data to be transferred within a process or between processes, and a systemwide mouse tracking service that can notify a process about the location and state of the mouse during a drag operation.

When a document is opened, the session object creates a single drag-and-drop object. All parts of that document share the drag-and-drop object; you can obtain a reference to it by calling the session object's `GetDragAndDrop` method (page 584). You must not cache this object; instead, you must call the session object's `GetDragAndDrop` method whenever you need the drag-and-drop object.

Data transfers requested by any part operate on the drag-and-drop object's content storage unit. You can obtain a reference to that storage unit by calling the drag-and-drop object's `GetContentStorageUnit` method (page 187). You must not cache that storage unit; instead you must call the `GetContentStorageUnit` method whenever you need to access the storage unit.

A drag initiated by an OpenDoc part moves or copies a single item, called a drag item. Some items that can be dragged are a content item, a text selection, or a selected embedded frame. A drag initiated by a non-OpenDoc application may have more than one drag item. For example, a drag initiated by an

application that manipulates the file system (such as the Finder on the Mac OS platform) might include several files.

When your part receives a mouse-down event that occurred within an item that can be dragged, you can initiate a drag. Initiating a drag involves acquiring the drag-and-drop object from the session object, clearing the drag-and-drop object's content storage unit, copying data from the item to be dragged into the content storage unit, and starting a drag action. Once the drag is initiated, the drag-and-drop object notifies a facet when the mouse passes over it. If the mouse button is released over a facet, the facet calls its part's `Drop` method (page 488) to notify the part of the drop. The destination part can retrieve the dragged data, using the supplied drag-item iterator. A drag-item iterator is an object of the `ODDragItemIterator` class (page 194); it allows the part to iterate through the drag items and obtain a reference to the content storage unit for each one.

For more information on drag-and-drop operations, see the chapter on data transfer in the *OpenDoc Programmer's Guide for the Mac OS*. For information on using a link specification to indicate that the source part can create a link, see the `ODLinkSpec` (page 379) class description.

Methods

This section presents summary descriptions of the `ODDragAndDrop` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [M] are specific to the Mac OS platform.

Initiating a Drag

`Clear`	Clears the content storage unit for this drag-and-drop object.
`StartDrag`	Initiates a drag operation.
`GetContentStorageUnit`	Returns a reference to the content storage unit for this drag-and-drop object.

Drag Information

`GetDragAttributes`	Returns additional information about the current drag-and-drop operation.

`GetDragReference` [M]
: Returns the Mac OS Drag Manager drag-reference number associated with the current drag operation.

User Interaction at Drop

`ShowPasteAsDialog` [M]
: Displays the Paste As dialog box and sets the appropriate dialog items according to the input parameters.

Clear

The `Clear` method clears the content storage unit for this drag-and-drop object.

```
void Clear ();
```

DISCUSSION

When your part initiates a drag, you must call this method immediately before calling the `GetContentStorageUnit` method. Doing so ensures that the content storage unit is empty and ready for drag-and-drop data transfers.

EXCEPTIONS

`kODErrNoDragManager`	No platform-specific drag system service is available.

SEE ALSO

The `ODDragAndDrop::GetContentStorageUnit` method (page 187).

GetContentStorageUnit

The `GetContentStorageUnit` method returns a reference to the content storage unit for this drag-and-drop object.

```
ODStorageUnit GetContentStorageUnit ();
```

return value A reference to the content storage unit.

DISCUSSION

When your part initiates a drag, you should first call the `Clear` method, then call this method and copy data from the drag item into the returned storage unit. The drag-and-drop object handles the creation and destruction of its content storage unit, so you must neither dispose of nor release the returned storage unit.

You must not cache the returned storage unit; instead you must call this method whenever you need to access the content storage unit.

EXCEPTIONS

`kODErrNoDragManager`	No platform-specific drag system service is available.

SEE ALSO

The `ODDragAndDrop::Clear` method (page 186).
The `ODStorageUnit` class (page 641).

GetDragAttributes

The `GetDragAttributes` method returns additional information about the current drag-and-drop operation.

```
ODULong GetDragAttributes ();
```

return value A 32-bit value consisting of flags that specify attributes of the current drag operation.

DISCUSSION

If no drag operation is in progress, the returned value is zero. Otherwise, the bit flags are set in the returned value to indicate attributes of the operation. For example, the `kODDragIsInSourcePart` flag is set if the drag has not left the source part; the `kODDropIsMove` flag is set if the drag items are being moved (not copied).

If a drag enters your part, or a drop occurs in your part, you can call this method to determine how your part should respond.

- If your part is tracking a drag that has entered one of its facets, you can decide whether to provide the user visual feedback by checking whether the `kODDragIsInSourcePart` or `kODDragIsInSourceFrame` flags are set in the returned value.
- If a drop occurred in your part, you can determine the appropriate response by checking which of the drop flags (for example, `kODDropIsMove` or `kODDropIsCopy`) are set in the returned value.

EXCEPTIONS

`kODErrNoDragManager`	No platform-specific drag system service is available.

SEE ALSO

"Drag Attributes" on page 891.

GetDragReference

Mac OS

The `GetDragReference` method returns the Mac OS Drag Manager drag reference number associated with the current drag operation.

```
ODPlatformDragReference GetDragReference ();
```

return value A 32-bit value identifying the current drag operation, or `kODNULL` if no drag operation is in progress.

DISCUSSION

You should call this method to obtain the reference number to use in direct calls to the Drag Manager, for example, to perform drag highlighting.

EXCEPTIONS

`kODErrNoDragManager`	No platform-specific drag system service is available.

ShowPasteAsDialog

Mac OS

The `ShowPasteAsDialog` method displays the Paste As dialog box and sets the appropriate dialog items according to the input parameters.

```
ODBoolean ShowPasteAsDialog (
                    in ODBoolean canPasteLink,
                    in ODPasteAsMergeSetting mergeSetting,
                    in ODFacet facet,
                    in ODTypeToken viewType,
                    in ODStorageUnit contentSU,
                    out ODPasteAsResult theResult);
```

`canPasteLink`
: `kODTrue` if the destination part allows a link to be created, otherwise `kODFalse`.

`mergeSetting`
: A value indicating whether embedding and merging are supported; determines the initial setting for the At the Destination radio buttons.

`facet`
: A reference to the facet in which the drop is to occur.

`viewType`
: A tokenized string representing the initial setting for the view type of the embedded part (if embedding is chosen).

`contentSU`
: A reference to the content storage unit for the drag item being pasted.

`theResult`
: A structure reflecting the user's selections in the Paste As dialog box.

return value
: `kODTrue` if the user clicked OK to leave the Paste As dialog box, otherwise `kODFalse`.

DISCUSSION

If your part is the destination of a drop and the user requests a Paste As operation, you can call this method to display the Paste As dialog box for a particular drag item. On the Mac OS platform, whenever your part is the destination of a drop, you should call the `GetDragAttributes` method of the drag-and-drop object; if the `kODDropIsPasteAs` flag is set in the drag attributes, you must call this method.

If the `canPasteLink` parameter is true, the content storage unit contains a link specification, and the draft permissions allow writing, then the Paste with Link checkbox is enabled. If that checkbox is enabled, its initial setting is checked.

The `mergeSetting` parameter specifies which At the Destination radio button (Merge with Contents or Embed As) is initially selected in the Paste As dialog box and whether the other button is available. It must be one of the following:

- Embed As is initially selected; Merge with Contents is available (`kODPasteAsEmbed`).
- Embed As is selected; Merge with Contents is disabled (`kODPasteAsEmbedOnly`).

- Merge with Contents is initially selected; Embed As is available (`kODPasteAsMerge`).
- Merge with Contents is selected; Embed As is disabled (`kODPasteAsMergeOnly`).

The `viewType` parameter must be the tokenized form of one of the view-type constants (`kODViewAsFrame`, `kODViewAsLargeIcon`, `kODViewAsSmallIcon`, or `kODViewAsThumbnail`). You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

If the user clicks OK, this method returns true and sets the fields of the output parameter, `theResult`, to indicate the selections the user made in the Paste As dialog box; in this case, you must deallocate the non-null `selectedKind`, `translateKind`, and `editor` fields of the `theResult` structure when you are finished using them.

If the user cancels the dialog box, this method returns false and you do not need to take any further action.

EXCEPTIONS

`kODErrIllegalNullStorageUnitInput`	The `ContentSU` parameter is null.
`kODErrNullFacetInput`	The `facet` parameter is null.
`kODErrNullPasteAsResultInput`	The `theResult` parameter is null.

SEE ALSO

The `ODPasteAsResult` type (page 888).
The `ODTypeToken` type (page 847).
The `ODDragAndDrop::GetDragAttributes` method (page 187).
The `ODSession::Tokenize` method (page 598).

StartDrag

The `StartDrag` method initiates a drag operation.

```
ODDropResult StartDrag (in ODFrame srcFrame,
                        in ODType imageType,
                        in ODByteArray image,
                        out ODPart destPart,
                        in ODByteArray refCon);
```

`srcFrame`	A reference to the frame in which the drag is initiated.
`imageType`	A platform-specific drag-image type indicating the type of image OpenDoc should display to the user as dragging feedback.
`image`	A byte array whose buffer contains the data of the image being dragged.
`destPart`	If the drop was successful, this parameter contains a reference to the destination part. Otherwise, the content of this parameter is undefined.
`refCon`	A byte array whose buffer contains extra platform-specific information needed for dragging.
return value	A value indicating the result of the drag-and-drop operation.

DISCUSSION

If your part initiates a drag, you should call this method after copying data from the drag item into the content storage unit for this drag-and-drop object. If the drag item includes any frames, you should call the `SetDragging` method for each frame to prevent it from being dragged into itself.

The content of the `image` parameter's buffer depends on the drag-image type. The only drag-image type supported on the Mac OS platform is a handle to a drag region (`kODDragImageRegionHandle`), as required by the Mac OS Drag Manager. For that drag-image type, the buffer contains the Mac OS `RgnHandle` value that is the handle of the drag region; the drag region is in global coordinates.

The data in the `refCon` parameter's buffer is platform-dependent. On the Mac OS, the buffer contains an `ODEventData` structure representing the mouse-down event that initiated the drag operation. This structure is needed by the Mac OS Drag Manager.

During the drag operation, the platform-specific drag manager displays the image specified in the `image` parameter and moves the image as the user moves the mouse pointer. After the user drops the image by releasing the mouse button, this method returns the result of the operation.

On the Mac OS, the returned value indicates a successful move (`kODDropMove`), a successful copy (`kODDropCopy`), or a failed drop (`kODDropFail`). On platforms that support asynchronous drag-and-drop operations, this method always returns `kODDropUnfinished`, indicating that the asynchronous operation has started.

When the drop is successful (the returned value is `kODDropMove` or `kODDropCopy`), the output parameter, `destPart`, contains a reference to the destination part; otherwise, the content of the `destPart` parameter is undefined.

EXCEPTIONS

`kODErrNoDragManager`	No platform-specific drag system service is available.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODDropResult` type (page 892)
The `ODEventData` type (page 860).
The `ODFrame::SetDragging` method (page 325).
The `ODPart::Drop` method (page 488).

ODDragItemIterator

Superclasses `ODObject`
Subclasses none

An object of the `ODDragItemIterator` class provides access to the content storage units for all drag items of a drag-and-drop object.

Description

Each drag item is an item being copied or moved in a drag-and-drop operation. Its content storage unit contains the data being transferred. Once a drag operation is initiated (by calling its drag-and-drop object's `StartDrag` method (page 192)), the drag-and-drop object notifies a part when the mouse pointer passes over one of its facets. In addition to providing the mouse location, the drag-and-drop object also supplies the part with a drag-item iterator. Similarly, if the mouse pointer is released over a frame, the part is notified of the mouse location and supplied with a drag-item iterator. In both cases, the part might use the drag-item iterator to examine each drag item from the incoming drag operation and decide whether to provide destination feedback and whether to accept the drop event.

While you are using a drag-item iterator, you should not modify the list of drag items. You must postpone adding items to or removing items from the list of drag items until after you have deleted the iterator.

For more information related to drag-and-drop objects, see the `ODDragAndDrop` class description (page 184). For more information on drag-and-drop operations, see the chapters on data transfer and OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*. For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODDragItemIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and returns a reference to the first storage unit in the iteration sequence.
`Next`	Returns a reference to the next storage unit in the iteration sequence.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and returns a reference to the first storage unit in the iteration sequence.

```
ODStorageUnit First ();
```

return value A reference to the first storage unit in the iteration sequence.

DISCUSSION

Your part must call this method before calling this drag-item iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

Because storage units are guaranteed to be valid only as long as the iterator is valid, you should never cache a reference to the returned storage unit. This method does not increment the reference count of the returned storage unit.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of drag items was modified while the iteration was in progress.

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more storage units remain in the iteration sequence. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found a storage unit. This method returns `kODFalse` when you have examined all the storage units.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The list of drag items was modified while the iteration was in progress.

Next

The `Next` method returns a reference to the next storage unit in the iteration sequence.

```
ODStorageUnit Next ();
```

return value A reference to the next storage unit in the iteration sequence, or `kODNULL` if you have reached the last storage unit.

DISCUSSION

If your part calls this method before calling this drag-item iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

Because storage units are guaranteed to be valid only as long as the iterator is valid, you should never cache a reference to the returned storage unit. This method does not increment the reference count of the returned storage unit.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of drag items was modified while the iteration was in progress.

ODEmbeddedFramesIterator

Superclasses `ODObject`

Subclasses none

An object of a subclass of `ODEmbeddedFramesIterator` provides access to all frames directly embedded within a display frame of your part.

Description

If your part is a container part, there is an additional class that you must subclass and implement along with your part editor (your subclass of `ODPart`). You must provide an embedded-frames iterator (a subclass of `ODEmbeddedFramesIterator`) to allow callers to access all frames directly embedded within a display frame of your part. A caller needs access to your part's embedded frames to access the parts embedded in those frames. For example, a caller might use an embedded-frames iterator if it has a spelling checker that needs to find all text parts in your document to operate.

The `ODEmbeddedFramesIterator` class is an abstract superclass that you must subclass to create your embedded-frames iterator. Callers create your embedded-frames iterator object by calling your part's `CreateEmbeddedFramesIterator` method (page 475), which returns a reference to an embedded-frames iterator object.

While you are using an embedded-frames iterator, you should not modify the list of embedded frames for the part. You must postpone adding frames to or removing frames from the list of embedded frames for the part until after you have deleted the iterator.

For more information related to frame objects, see the `ODFrame` class description (page 288). For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODEmbeddedFramesIterator`.

somInit

The `somInit` method initializes the instance variables in a SOM object; it is inherited from the `SOMObject` class.

When you subclass `ODEmbeddedFramesIterator`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in this embedded-frames iterator object. The SOM library calls this method when this embedded-frames iterator is created. You must not do anything that might fail in this method. This limits you to operations like setting pointer variables to null, setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in your embedded-frames iterator's `InitEmbeddedFramesIterator` method (page 203).

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

When you subclass `ODEmbeddedFramesIterator`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for this embedded-frames iterator object, including any storage related to additional instance variables initialized in this embedded-frames iterator object. The SOM library calls this method when this embedded-frames iterator is deleted; this method must not fail.

Purge

The `Purge` method frees memory on request; it is inherited from the `ODObject` class.

```
ODSize Purge (in ODSize size);
```

Every subclass of `ODObject` can override this method and should do so if it creates caches and temporary buffers. When you subclass `ODEmbeddedFramesIterator`, you must override this method or risk running out of available memory. Your override method must call its inherited method at some point in your implementation (it does not matter where). You should save the size value returned by the inherited method because you will need it to compute the value to return from your override method.

Your override of this method should free any caches, noncritical buffers, or objects (up to the amount of memory specified). Your override of this method should add the number of bytes actually freed to the number returned by the inherited method and return the sum as the total amount of memory released. OpenDoc calls this method in low-memory situations; you should not allocate memory for this operation.

Methods

This section presents summary descriptions of the `ODEmbeddedFramesIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Initializing

`InitEmbeddedFramesIterator` Should initialize this embedded-frames iterator object.

Accessing

`First` Should begin the iteration and return a reference to the first frame in the iteration sequence.

`Next` Should return a reference to the next frame in the iteration sequence.

Iterator Testing

`IsNotComplete`	Should return a Boolean value that indicates whether the iteration is incomplete.
`CheckValid`	Should check whether this embedded-frames iterator object is valid and generates an exception if it is not valid.
`IsValid`	Should return a Boolean value that indicates whether this embedded-frames iterator is valid.
`PartRemoved`	Should invalidate this embedded-frames iterator.

CheckValid

The `CheckValid` method should check whether this embedded-frames iterator object is valid and generates an exception if it is not valid.

```
void CheckValid ();
```

DISCUSSION

Every subclass of `ODEmbeddedFramesIterator` must test the embedded-frames iterator's validity at the beginning of the implementation of each of its noninherited methods (except for the subclass-specific initialization method) by calling either this method or the `IsValid` method. Unlike the `IsValid` method, this method has no effect if this embedded-frames iterator is valid; otherwise it should generate an exception.

If you want to ensure that you make calls only to a valid embedded-frames iterator, without generating an exception, then call the `IsValid` method instead.

OVERRIDING

When you subclass `ODEmbeddedFramesIterator`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidIterator`
: This embedded-frames iterator is invalid and should not be used because the part that created it no longer exists.

SEE ALSO

The `ODEmbeddedFramesIterator::IsValid` method (page 205).

First

The `First` method should begin the iteration and return a reference to the first frame in the iteration sequence.

```
ODFrame First ();
```

return value A reference to the first frame in the iteration sequence, or `kODNULL` if the part has no embedded frames.

DISCUSSION

A client of this embedded-frames iterator calls this method before calling this embedded-frames iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

Your override of this method should not increment the reference count of the returned frame object.

OVERRIDING

When you subclass `ODEmbeddedFramesIterator`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of embedded frames for the part was modified while the iteration was in progress.

InitEmbeddedFramesIterator

The `InitEmbeddedFramesIterator` method should initialize this embedded-frames iterator object.

```
void InitEmbeddedFramesIterator (in ODPart part);
```

`part` A reference to a part whose frames this iterator traverses.

DISCUSSION

Your part's `CreateEmbeddedFramesIterator` method calls this method when this embedded-frames iterator is created. A client that uses this embedded-frames iterator does not call this method.

OVERRIDING

When you subclass `ODEmbeddedFramesIterator`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODPart::CreateEmbeddedFramesIterator` method (page 475).

IsNotComplete

The `IsNotComplete` method should return a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

A client of this embedded-frames iterator calls this method to test whether more frames remain in the iteration sequence. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found a frame. This method returns `kODFalse` when you have examined all the frames.

OVERRIDING

When you subclass `ODEmbeddedFramesIterator`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The list of embedded frames for the part was modified while the iteration was in progress.

IsValid

The `IsValid` method should return a Boolean value that indicates whether this embedded-frames iterator is valid.

```
ODBoolean IsValid ();
```

return value `kODTrue` if this embedded-frames iterator is valid, otherwise `kODFalse`.

DISCUSSION

A client of this embedded-frames iterator calls this method if it wants to ensure that it makes calls only to a valid embedded-frames iterator, without generating an exception.

OVERRIDING

When you subclass `ODEmbeddedFramesIterator`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODEmbeddedFramesIterator::CheckValid` method (page 201).

Next

The `Next` method should return a reference to the next frame in the iteration sequence.

```
ODFrame Next ();
```

return value A reference to the next frame in the iteration sequence, or `kODNULL` if you have reached the last frame.

DISCUSSION

A client of this embedded-frames iterator calls this method. If the client calls this method before calling this embedded-frames iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

Your override of this method should not increment the reference count of the returned frame object.

OVERRIDING

When you subclass `ODEmbeddedFramesIterator`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of embedded frames for the part was modified while the iteration was in progress.

PartRemoved

The `PartRemoved` method should invalidate this embedded-frames iterator.

```
void PartRemoved ();
```

DISCUSSION

The part whose embedded frames this iterator traverses calls this method when the part closes. This embedded-frames iterator then becomes invalid and should no longer attempt to communicate with its part; any pointers that this embedded-frames iterator had to its part should be considered invalid.

OVERRIDING

When you subclass `ODEmbeddedFramesIterator`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

ODExtension

Superclasses `ODRefCntObject` → `ODObject`

Subclasses `ODSemanticInterface` and `ODSettingsExtension`

An object of the `ODExtension` class represents an extended interface to an OpenDoc object.

Description

The OpenDoc architecture is designed to be extended. You can enhance the capabilities of and communications among your parts or other OpenDoc objects in a compound document by extending the standard OpenDoc interfaces. All subclasses of `ODObject` (including shapes, facets, frames, documents, windows, and parts) can be extended. A part editor can define an extended interface for any purpose, including extensions to handle text searching, linking, specialized text formatting, database accessing, and specialized graphics processing. You can even use extensions to develop component software that goes well beyond the standard OpenDoc model of parts and compound documents.

The `ODExtension` class is an abstract superclass that you can subclass to create an extended interface to a base object. For example, the `ODSemanticInterface` class (page 557) is a subclass of `ODExtension`. Callers can access an already existing extension object by calling its base object's `AcquireExtension` method (page 426), which returns a reference to the extension object.

The `ODExtension` class itself has minimal functionality. Each extension object knows which object it is an extension of and how to release resources in itself and in its base object. Further behavior should be implemented in a subclass of `ODExtension`.

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODExtension`.

somInit

The `somInit` method initializes the instance variables in a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODExtension`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in this extension object. The SOM library calls this method when this extension object is created. You must not do anything that might fail in this method. This limits you to operations like setting pointer variables to null, setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in this extension's subclass-specific initialization method; see also the `InitExtension` method (page 213).

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODExtension`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for this extension object, including any storage related to additional instance variables initialized in this extension object. The SOM library calls this method when this extension object is deleted; this method must not fail.

Release

The `Release` method decrements an object's reference count by 1; it is inherited from the `ODRefCntObject` class.

```
void Release ();
```

If you subclass `ODExtension`, you can override this method to release an object and reclaim valuable resources like memory. Your override method must call its inherited method at the beginning of your implementation.

A part editor calls this method when it no longer needs a reference to this extension object. If this extension object's reference count becomes 0 and the base object is not null, the base object's `ReleaseExtension` method is called.

Purge

The `Purge` method frees memory on request; it is inherited from the `ODObject` class.

```
ODSize Purge (in ODSize size);
```

Every subclass of `ODObject` can override this method and should do so if it creates caches and temporary buffers. If you subclass `ODExtension`, you must override this method or risk running out of available memory. Your override method must call its inherited method at some point in your implementation (it does not matter where). You should save the size value returned by the inherited method because you will need it to compute the value to return from your override method.

Your override of this method should free any caches, noncritical buffers, or objects (up to the amount of memory specified). Your override of this method should add the number of bytes actually freed to the number returned by the inherited method and return the sum as the total amount of memory released. OpenDoc calls this method in low-memory situations; you should not allocate memory for this operation.

Methods

This section presents summary descriptions of the `ODExtension` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Initializing

`InitExtension`	Initializes this extension object.

Base-Object Manipulation

`GetBase`	Returns a reference to this extension's base object.
`BaseRemoved`	Invalidates this extension object.

Extension Characteristics

`CheckValid`	Checks whether this extension object is valid and generates an exception if it is not valid.
`IsValid`	Returns a Boolean value that indicates whether this extension object is valid.

BaseRemoved

The `BaseRemoved` method invalidates this extension object.

```
void BaseRemoved ();
```

DISCUSSION

An extension object becomes invalid when its base object is removed. This extension object should no longer attempt to communicate with its base object; any pointers that this extension object had to its base object should be considered invalid. The base object of this extension object should call this method from its `ReleaseAll` method.

OVERRIDING

If you subclass `ODExtension`, you can override this method. Your override method must call its inherited method at some point in your implementation (it does not matter where); you must not access the base object after calling the inherited method.

CheckValid

The `CheckValid` method checks whether this extension object is valid and generates an exception if it is not valid.

```
void CheckValid ();
```

DISCUSSION

Every subclass of `ODExtension` must test the extension object's validity at the beginning of the implementation of each of its noninherited methods (except for the subclass-specific initialization method) by calling either this method or the `IsValid` method. Unlike the `IsValid` method, calling this method has no effect if this extension object is valid; otherwise it generates an exception.

If you want to ensure that you make calls only to a valid extension object, without generating an exception, then call the `IsValid` method instead.

EXCEPTIONS

`kODErrInvalidExtension`
: This extension object is invalid and should not be used because its base object no longer exists.

SEE ALSO

The `ODExtension::IsValid` method (page 214).

GetBase

The `GetBase` method returns a reference to this extension's base object.

```
ODObject GetBase ();
```

return value A reference to this extension's base object.

DISCUSSION

A client of this extension object calls this method if it has a reference to this extension object, but did not save a reference to its base object.

InitExtension

The `InitExtension` method initializes this extension object.

```
void InitExtension (in ODObject base);
```

`base` A reference to this extension's base object.

DISCUSSION

This method is not called directly to initialize this extension object, but is called by a subclass-specific initialization method. By convention, every subclass of `ODExtension` should have a separate initialization method (for example, the `InitMyExtension` method) that is called when an instance of that subclass is created. The override method may have additional parameters beyond those of the `InitExtension` method. The `InitMyExtension` method should call the inherited `InitExtension` method at the beginning of its implementation.

If you subclass `ODExtension`, your subclass-specific initialization method, rather than its `somInit` method, should handle any initialization code that can potentially fail. For example, your initialization method may attempt to allocate memory for your extension.

OVERRIDING

If you subclass `ODExtension`, you should not override this method.

IsValid

The `IsValid` method returns a Boolean value that indicates whether this extension object is valid.

```
ODBoolean IsValid ();
```

return value `kODTrue` if this extension object is valid, otherwise `kODFalse`.

DISCUSSION

A client of this extension object calls this method if it wants to ensure that it makes calls only to a valid extension object, without generating an exception.

SEE ALSO

The `ODExtension::CheckValid` method (page 212).

ODFacet

Superclasses `ODObject`

Subclasses none

An object of the `ODFacet` class holds nonpersistent information about the layout of parts and describes the location of a frame on a particular canvas for display and event dispatching.

Description

A facet object represents a visible, drawable frame or portion of a frame. In the simplest case, each embedded frame in a document is displayed and manipulated using a single facet. In some cases, however, your part might need to display portions of the same embedded frame in several places, such as in a split view. In that case, one or more facets might manage the layout of one frame. All facets of a frame might display that frame's content identically, or a part might store user-defined part info data in the facet to distinguish among its different facets. The part info might also consist of graphics-system-specific information used for displaying the facet, such as a QuickDraw GX view port.

Facets are organized hierarchically. Each window or printed page has a single facet, the **root facet**, which is the topmost facet visible in the document window. All other facets contained in that window descend from the root facet. Each facet of a given frame is contained within a facet of that frame's containing frame.

Your part creates a facet object for each visible embedded frame by calling its own display facet's `CreateEmbeddedFacet` method (page 234). Your part can also create a root facet (for printing) by calling the window-state object's `CreateFacet` method (page 828). These methods return a reference to a facet object.

Facets may or may not exist for frames that have been previously visible and that have scrolled out of view, and parts may or may not create facets for frames that are expected to be visible soon. The only requirement is that facets

must exist for all currently visible frames. A visible facet always has a frame, and that frame is defined for the lifetime of the facet; once set, it cannot be changed. Facets that are not currently visible in a window may be purged from memory under low-memory conditions.

Facet Geometry

Facets hold information regarding the geometry of their corresponding frames. Each facet maintains an active shape, a clip shape, and an external transform. The clip shape and external transform must always be valid.

- The **active shape** defines the area within a frame in which the facet's embedded part is willing to receive geometry-based user events, such as mouse clicks. It is commonly identical to the frame shape of the facet's frame, but it might be modified to coincide with the used shape or some other shape. The facet's embedded part controls the active shape.
- The **clip shape** defines the area within a frame in which drawing can occur; it is the area unobscured by overlapping content of the containing part. If it is unobscured, the clip shape is identical to the frame shape of the facet's frame. The facet's containing part controls the clip shape.
- The **external transform** describes the transform that is applied to a facet to position, scale, or otherwise transform the image drawn within the facet in the coordinate space of its containing part. The facet's containing part controls the external transform.

Facets and Canvases

A facet might possess its own canvas. If a particular facet in a window's facet hierarchy has an attached canvas, it and all its embedded facets (and their embedded facets, and so on) draw to that canvas. Each facet inherits its canvas from its containing facet; the inherited canvas is called the parent canvas. For most drawing, only a window's root facet needs a canvas. For offscreen double-buffering or image manipulation, however, a part can create a canvas and attach it to an embedded facet, copying the image of the offscreen canvas to its parent canvas during updates.

Because a facet can simultaneously have both a window canvas and an offscreen canvas, there can be two drawing environments to consider. The aggregate clip shape, content transform, and frame transform position and clip

the facet on the closest drawing environment (the closest canvas in the facet hierarchy, which may or may not be the window canvas). During real-time interactions such as rubber-banding or dragging, your part might need to display directly in the window. In such cases, the window aggregate clip shape, window content transform, and window frame transform specifically position and clip the facet on the window canvas.

The `ODFacet` class includes several methods that specify geometry (shape and transform objects) or calculate positions on a canvas. Because these calculations necessarily assume a coordinate system, the `ODFacet` methods include a parameter, `biasCanvas`, that allows you to specify a canvas to whose coordinate space the geometry is biased. The bias canvas uses a bias transform to convert from the coordinate system used for drawing on the canvas to the coordinate system (platform-normal coordinates) used by the current graphics system. For more information related to bias transforms, see the `ODCanvas` class description (page 61).

Methods

This section presents summary descriptions of the `ODFacet` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Creating Objects

`CreateEmbeddedFacet`
: Creates a facet for the specified frame embedded in this facet.

`CreateFacetIterator`
: Creates a facet iterator object for the embedded facets of this facet.

`CreateCanvas`
: Creates a canvas object.

`CreateShape`
: Creates a shape object.

`CreateTransform`
: Creates a transform object.

Facet Hierarchy

`GetContainingFacet`
: Returns a reference to the containing facet of this facet.

`GetFrame`
: Returns a reference to this facet's frame.

`GetWindow`	Returns a reference to the window this facet is displayed in.
`RemoveFacet`	Removes an embedded facet from this facet.
`MoveBefore`	Repositions an embedded facet of this facet in front of a sibling facet.
`MoveBehind`	Repositions an embedded facet of this facet behind a sibling facet.

Facet Geometry

`AcquireActiveShape`	Returns a reference to the active shape for this facet.
`ChangeActiveShape`	Assigns the specified shape as the active shape of this facet.
`AcquireClipShape`	Returns a reference to this facet's clip shape.
`ChangeGeometry`	Assigns the specified clip shape, external transform, or both to this facet.
`AcquireExternalTransform`	Returns a reference to this facet's external transform.

Facet-Canvas Geometry

`GetCanvas`	Returns a reference to the canvas associated with this facet.
`ChangeCanvas`	Attaches a canvas to this facet.
`HasCanvas`	Returns a Boolean value that indicates whether this facet has its own canvas.
`AcquireAggregateClipShape`	Calculates and returns a reference to a shape object that represents the aggregate clip shape of this facet.
`AcquireContentTransform`	Calculates and returns a reference to a transform object that represents the content transform of this facet.
`AcquireFrameTransform`	Calculates and returns a reference to a transform object that represents the frame transform of this facet.

Window-Canvas Geometry

`AcquireWindowAggregateClipShape`
: Calculates and returns a reference to a shape object that represents the window aggregate clip shape of this facet.

`AcquireWindowContentTransform`
: Calculates and returns a reference to a transform object that represents the window-content transform of this facet.

`AcquireWindowFrameTransform`
: Calculates and returns a reference to a transform object that represents the window-frame transform of this facet.

Point Testing

`ContainsPoint`
: Returns a Boolean value that indicates whether the specified point is within the area of this facet.

`ActiveBorderContainsPoint`
: Returns a Boolean value that indicates whether this facet's frame is active and a point is within its active frame border.

Imaging

`IsSelected`
: Returns a Boolean value that indicates whether this facet is selected within its containing part.

`SetSelected`
: Specifies whether this facet is currently being selected within its containing part.

`GetHighlight`
: Returns the highlight state for this facet.

`ChangeHighlight`
: Changes the highlight state for this facet.

`Draw`
: Tells this facet's part to draw itself within the specified portion of this facet.

`DrawActiveBorder`
: Updates the active frame border for this facet.

`DrawChildren`
: Draws all embedded facets of this facet that need updating.

`DrawChildrenAlways`
: Draws all embedded facets of this facet, whether they need updating or not.

`DrawnIn`	Called when this facet has been drawn in without an update event being generated.
`Invalidate`	Marks the specified area in this facet as in need of updating.
`InvalidateActiveBorder`	Marks the active frame border of this facet as in need of updating.
`Validate`	Marks the specified area in this facet as no longer in need of updating.
`Update`	Updates this facet's canvas by drawing this facet and any of its embedded facets whose clip shape intersects the specified area of the canvas.

Part Info

`GetPartInfo`	Returns the part info data for this facet.
`SetPartInfo`	Assigns part info data to this facet.

AcquireActiveShape

The `AcquireActiveShape` method returns a reference to the active shape for this facet.

```
ODShape AcquireActiveShape (in ODCanvas biasCanvas);
```

`biasCanvas`	`kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.
return value	A reference to the active shape for this facet, or if no active shape has been set, the frame shape of this facet's frame.

DISCUSSION

This method increments the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

SEE ALSO

The `ODFacet::ChangeActiveShape` method (page 229).

AcquireAggregateClipShape

The `AcquireAggregateClipShape` method calculates and returns a reference to a shape object that represents the aggregate clip shape of this facet.

```
ODShape AcquireAggregateClipShape (in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to a shape object that represents the aggregate clip shape.

DISCUSSION

The aggregate clip shape describes this facet's clip shape on its canvas. The aggregate clip shape is calculated by intersecting this facet's clip shape with the appropriately transformed clip shapes of all containing facets displayed on this facet's canvas.

This method increments the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

SEE ALSO

The `ODFacet::AcquireClipShape` method (page 222).
The `ODFacet::AcquireWindowAggregateClipShape` method (page 225).

AcquireClipShape

The `AcquireClipShape` method returns a reference to this facet's clip shape.

```
ODShape AcquireClipShape (in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to this facet's clip shape.

DISCUSSION

This method increments the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

SEE ALSO

The `ODFacet::AcquireAggregateClipShape` method (page 221).
The `ODFacet::AcquireWindowAggregateClipShape` method (page 225).

AcquireContentTransform

The `AcquireContentTransform` method calculates and returns a reference to a transform object that represents the content transform of this facet.

```
ODTransform AcquireContentTransform (
                                    in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to a transform object that represents the content transform.

DISCUSSION

The content transform describes the content coordinate space of this facet on its canvas. The content transform is calculated by concatenating the internal transform of this facet's frame with this facet's external transform and the internal and external transforms of all containing facets displayed on this facet's canvas.

This method increments the reference count of the returned transform object. When you have finished using that transform object, you should call its `Release` method.

SEE ALSO

The `ODFacet::AcquireExternalTransform` method (page 223).
The `ODFacet::AcquireFrameTransform` method (page 224).
The `ODFacet::AcquireWindowContentTransform` method (page 226).

AcquireExternalTransform

The `AcquireExternalTransform` method returns a reference to this facet's external transform.

```
ODTransform AcquireExternalTransform (
                                    in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to this facet's external transform.

DISCUSSION

This method increments the reference count of the returned transform object. When you have finished using that transform object, you should call its `Release` method.

SEE ALSO

The `ODFacet::AcquireContentTransform` method (page 222).
The `ODFacet::AcquireFrameTransform` method (page 224).

AcquireFrameTransform

The `AcquireFrameTransform` method calculates and returns a reference to a transform object that represents the frame transform of this facet.

```
ODTransform AcquireFrameTransform (in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to a transform object that represents the frame transform.

DISCUSSION

The frame transform describes the frame coordinate space of this facet on its canvas. The frame transform is calculated by concatenating this facet's external transform with the internal and external transforms of all containing facets displayed on this facet's canvas.

This method increments the reference count of the returned transform object. When you have finished using that transform object, you should call its `Release` method.

SEE ALSO

The `ODFacet::AcquireContentTransform` method (page 222).
The `ODFacet::AcquireExternalTransform` method (page 223).
The `ODFacet::AcquireWindowFrameTransform` method (page 227).

AcquireWindowAggregateClipShape

The `AcquireWindowAggregateClipShape` method calculates and returns a reference to a shape object that represents the window aggregate clip shape of this facet.

```
ODShape AcquireWindowAggregateClipShape (
                                    in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to a shape object that represents the window aggregate clip shape.

DISCUSSION

The window aggregate clip shape describes this facet's clip shape on its window. The window aggregate clip shape is calculated by intersecting the clip shape of this facet with the clip shapes of all containing facets displayed on this facet's window canvas.

This method increments the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

SEE ALSO

The `ODFacet::AcquireAggregateClipShape` method (page 221).
The `ODFacet::AcquireClipShape` method (page 222).

AcquireWindowContentTransform

The `AcquireWindowContentTransform` method calculates and returns a reference to a transform object that represents the window-content transform of this facet.

```
ODTransform AcquireWindowContentTransform (
                                    in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to a transform object that represents the window-content transform.

DISCUSSION

The window-content transform describes the content coordinate space of this facet on its window. The window-content transform is calculated by concatenating the internal transform of this facet's frame with this facet's external transform and the internal and external transforms of all containing facets displayed on this facet's window canvas.

This method increments the reference count of the returned transform object. When you have finished using that transform object, you should call its `Release` method.

SEE ALSO

The `ODFacet::AcquireContentTransform` method (page 222).
The `ODFacet::AcquireWindowFrameTransform` method (page 227).

AcquireWindowFrameTransform

The `AcquireWindowFrameTransform` method calculates and returns a reference to a transform object that represents the window-frame transform of this facet.

```
ODTransform AcquireWindowFrameTransform (
                                    in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to a transform object that represents the window-frame transform.

DISCUSSION

The window-frame transform describes the frame coordinate space of this facet on its window. The window-frame transform is calculated by concatenating this facet's external transform with the internal and external transforms of all containing facets displayed on this facet's window canvas.

This method increments the reference count of the returned transform object. When you have finished using that transform object, you should call its `Release` method.

SEE ALSO

The `ODFacet::AcquireFrameTransform` method (page 224).
The `ODFacet::AcquireWindowContentTransform` method (page 226).

ActiveBorderContainsPoint

The `ActiveBorderContainsPoint` method returns a Boolean value that indicates whether this facet's frame is active and a point is within its active frame border.

```
ODBoolean ActiveBorderContainsPoint (
                                    in ODPoint point,
                                    in ODCanvas biasCanvas);
```

`point`	The location to test, expressed in frame coordinates.
`biasCanvas`	`kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.
return value	`kODTrue` if this facet's frame is active and a point is within its active frame border, otherwise `kODFalse`.

DISCUSSION

Your part calls its embedded facet's `ActiveBorderContainsPoint` method when it receives a mouse-enter or mouse-within event. Your part uses this method to do mouse-up event tracking by checking to see if the event received is in one of its embedded facets.

OpenDoc draws this facet's active frame border on the border of the active shape associated with this facet. The active frame border is derived from the active shape, but is not identical to it.

SEE ALSO

The `ODPoint` type (page 855).
The `ODFacet::ContainsPoint` method (page 232).

ChangeActiveShape

The `ChangeActiveShape` method assigns the specified shape as the active shape of this facet.

```
void ChangeActiveShape (in ODShape activeShape,
                        in ODCanvas biasCanvas);
```

`activeShape` A reference to a shape to assign as this facet's active shape.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

When this facet's frame has the selection focus, OpenDoc automatically takes care of changing the facet's active border shape and redrawing the active frame border. Otherwise, only this facet's part should change this facet's active shape.

SEE ALSO

The `ODFacet::AcquireActiveShape` method (page 220).

ChangeCanvas

The `ChangeCanvas` method attaches a canvas to this facet.

```
void ChangeCanvas (in ODCanvas canvas);
```

`canvas` A reference to a canvas to attach to this facet.

DISCUSSION

This method in turn calls its canvas's `SetFacet` method to ensure the canvas has a reference back to this facet. After this method executes successfully, you can display this facet and its embedded facets on the specified canvas.

Mac OS
It is the caller's responsibility to deallocate the canvas's storage when it is no longer needed. ◆

EXCEPTIONS

`kODErrCanvasHasNoOwner`
The offscreen canvas has no owner.

SEE ALSO

The `ODCanvas::SetFacet` method (page 77).
The `ODFacet::GetCanvas` method (page 241).
The `ODFacet::HasCanvas` method (page 245).

ChangeGeometry

The `ChangeGeometry` method assigns the specified clip shape, external transform, or both to this facet.

```
void ChangeGeometry (in ODShape clipShape,
                     in ODTransform transform,
                     in ODCanvas biasCanvas);
```

`clipShape` A reference to a shape to assign as this facet's clip shape.

`transform` A reference to a transform to assign as this facet's external transform.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

This method invalidates any cached aggregate clip shape and transforms. This facet's containing part calls this method to change this facet's clip shape, external transform, or both. This method in turn calls the `GeometryChanged` method associated with this facet's part to notify the part (and the parts of all its embedded facets) that its clip shape, external transform, or both has changed.

SEE ALSO

The `ODPart::GeometryChanged` method (page 503).

ChangeHighlight

The `ChangeHighlight` method changes the highlight state for this facet.

```
void ChangeHighlight (in ODHighlight highlight);
```

`highlight` The possible highlight state to assign to this facet. The value of `highlight` must be one of the following: `kODNoHighlight`, `kODFullHighlight`, or `kODDimHighlight`.

DISCUSSION

This facet's containing part calls this method to change this facet's highlight state (usually so that its embedded part's highlighting corresponds to that of its containing part). This method in turn calls the `HighlightChanged` method associated with this facet's part to notify the part that its highlight state has changed.

SEE ALSO

The `ODHighlight` type (page 850).
The `ODFacet::GetHighlight` method (page 243).
The `ODPart::HighlightChanged` method (page 508).

ContainsPoint

The `ContainsPoint` method returns a Boolean value that indicates whether the specified point is within the area of this facet.

```
ODBoolean ContainsPoint (in ODPoint point,
                         in ODCanvas biasCanvas);
```

`point`	The location to test, expressed in frame coordinates.
`biasCanvas`	`kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.
return value	`kODTrue` if the specified point is within the area of this facet, otherwise `kODFalse`.

DISCUSSION

Your part calls its embedded facet's `ContainsPoint` method when it needs to do hit-testing on those facets. This method tests the result against the intersection of the clip shape and the active shape.

SEE ALSO

The `ODPoint` type (page 855).
The `ODFacet::ActiveBorderContainsPoint` method (page 228).

CreateCanvas

The `CreateCanvas` method creates a canvas object.

```
ODCanvas CreateCanvas (in ODGraphicsSystem graphicsSystem,
                       in ODPlatformCanvas platformCanvas,
                       in ODBoolean isDynamic,
                       in ODBoolean isOffscreen);
```

`graphicsSystem`
: A 16-bit value specifying the graphics system you want to use for the canvas. Valid values for `graphicsSystem` are platform-dependent.

`platformCanvas`
: A 32-bit value identifying the graphics-system-specific drawing structure to assign to the canvas, or `kODNULL` for no drawing structure. Valid values for `platformCanvas` are graphics-system-dependent.

`isDynamic` `kODTrue` if this canvas is to be dynamic, otherwise `kODFalse`.

`isOffscreen`
: `kODTrue` if this canvas is to be offscreen, otherwise `kODFalse`.

return value A reference to a new canvas object.

DISCUSSION

Your part calls this method to create an offscreen canvas object to attach to this facet. To create a canvas that will not be attached to any facet, call the window-state object's `CreateCanvas` method.

On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`). For QuickDraw, the platform canvas should be a QuickDraw graphics port (type `GrafPtr`); for QuickDraw GX, it should be a QuickDraw GX view port object (type `gxViewPort`).

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODWindowState::CreateCanvas` method (page 827).
The `ODCanvas` class (page 61).

CreateEmbeddedFacet

The `CreateEmbeddedFacet` method creates a facet for the specified frame embedded in this facet.

```
ODFacet CreateEmbeddedFacet (
                        in ODFrame frame,
                        in ODShape clipShape,
                        in ODTransform externalTransform,
                        in ODCanvas canvas,
                        in ODCanvas biasCanvas,
                        in ODFacet siblingFacet,
                        in ODFramePosition position);
```

`frame` A reference to a frame for this facet.

`clipShape` A reference to an initial clip shape for this facet.

`externalTransform`
A reference to an initial external transform for this facet.

`canvas` A reference to a canvas this facet should draw to, or `kODNULL` if identical to the canvas associated with the containing facet.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

`siblingFacet`
A reference to an existing embedded facet of this facet, or `kODNULL` if the embedded facet does not exist.

`position` The desired position of the new facet relative to the sibling facet. The value of `position` must be one of the following: `kODFrameInFront` or `kODFrameBehind`. If the sibling facet is set to `kODNULL`, the new facet is placed in front of (`kODFrameInFront`) or behind all (`kODFrameBehind`) its sibling facets.

return value A reference to a new facet object.

DISCUSSION

This facet's part calls this method to create a facet for one of its embedded frames. This method in turn calls its frame's `FacetAdded` method, which in turn calls the `FacetAdded` method associated with the frame's part to notify the part that a facet has been added to one of its display frames.

EXCEPTIONS

`kODErrCanvasHasNoOwner`
The offscreen canvas has no owner.

`kODErrInvalidFacet` The specified sibling facet is not an embedded facet of this facet.

`kODErrUnsupportedFramePositionCode`
The specified position is not a valid position code.

SEE ALSO

The `ODFramePosition` type (page 852).
The `ODFrame::FacetAdded` method (page 312).
The `ODPart::FacetAdded` method (page 496).

CreateFacetIterator

The `CreateFacetIterator` method creates a facet iterator object for the embedded facets of this facet.

```
ODFacetIterator CreateFacetIterator (
                        in ODTraversalType traversalType,
                        in ODSiblingOrder siblingOrder);
```

`traversalType`
The traversal type to assign to the facet iterator. The value of `traversalType` must be one of the following: `kODTopDown`, `kODBottomUp`, or `kODChildrenOnly`.

`siblingOrder`
: The order (either front-to-back or back-to-front) in which the iterator traverses the sibling facets. The value of `siblingOrder` must be one of the following: `kODBackToFront` or `kODFrontToBack`.

return value
: A reference to a new facet iterator object.

DISCUSSION

The value `kODTopDown` for the `traversalType` parameter indicates that you should traverse the facet hierarchy top down, in depth-first order, with this facet as the root. The value `kODBottomUp` indicates that you should traverse the facet hierarchy bottom up, visiting this facet after visiting all its children. Both `kODTopDown` and `kODBottomUp` include this facet in the traversal. The value `kODChildrenOnly` indicates that you should traverse only the children of this facet (not including this facet itself).

Your part calls this method if it needs to apply an operation to a facet and all its embedded facets. For example, OpenDoc might use a facet iterator to make sure that all facets within a facet being invalidated are also invalidated. It is your responsibility to delete the iterator when it is no longer needed.

While you are using a facet iterator, you should not modify the list of embedded facets. You must postpone adding items to or removing items from the list of embedded facets until after you have deleted the iterator.

SEE ALSO

The `ODTraversalType` type (page 851).
The `ODSiblingOrder` type (page 851).
The `ODFrame::CreateFacetIterator` method (page 308).
The `ODFacetIterator` class (page 253).

CreateShape

The `CreateShape` method creates a shape object.

```
ODShape CreateShape ();
```

return value A reference to a new shape object.

DISCUSSION

Your part calls this method to create a shape object for any purpose.

This method initializes the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

SEE ALSO

The `ODFrame::CreateShape` method (page 309).
The `ODShape` class (page 606).

CreateTransform

The `CreateTransform` method creates a transform object.

```
ODTransform CreateTransform ();
```

return value A reference to a new transform object.

DISCUSSION

Your part calls this method to create a transform object for any purpose.

This method initializes the reference count of the returned transform object. When you have finished using that transform object, you should call its `Release` method.

SEE ALSO

The `ODFrame::CreateTransform` method (page 237).
The `ODTransform` class (page 736).

Draw

The `Draw` method tells this facet's part to draw itself within the specified portion of this facet.

```
void Draw (in ODShape invalidShape,
           in ODCanvas biasCanvas);
```

`invalidShape`
A reference to a shape within which the part should draw itself, expressed in frame coordinates.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

This facet's containing part calls this method when the embedded facets of this facet require updating. This method in turn calls the `Draw` method associated with this facet's part.

SEE ALSO

The `ODPart::Draw` method (page 487).
The `ODFacet::DrawChildren` method (page 239).
The `ODFacet::DrawChildrenAlways` method (page 240).

DrawActiveBorder

The `DrawActiveBorder` method updates the active frame border for this facet.

```
void DrawActiveBorder ();
```

DISCUSSION

OpenDoc calls this method. When this facet's frame has the selection focus, OpenDoc automatically takes care of changing the facet's active border shape and redrawing the active frame border.

Under normal circumstances, there is no need to invalidate the active frame border for either the active part or its containing part. However, parts could call this method if they want to force the active frame border to be redrawn when they change the shape of an embedded part. To do so, the part must invalidate the active frame border by calling its facet's `InvalidateActiveBorder` method and redraw the active frame border by calling its facet's `DrawActiveBorder` method.

SEE ALSO

The `ODFacet::InvalidateActiveBorder` method (page 246).
The `ODFrame::DrawActiveBorder` method (page 311).

DrawChildren

The `DrawChildren` method draws all embedded facets of this facet that need updating.

```
void DrawChildren (in ODShape invalidShape,
                   in ODCanvas biasCanvas);
```

`invalidShape`
A reference to a shape within which the embedded facets should draw, expressed in the frame coordinates of this facet.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

This facet's containing part calls this method when the embedded facets of this facet require updating. This method in turn calls the `Draw` method associated with this facet's embedded part.

SEE ALSO

The `ODFacet::DrawChildrenAlways` method (page 240).
The `ODPart::Draw` method (page 487).

DrawChildrenAlways

The `DrawChildrenAlways` method draws all embedded facets of this facet, whether they need updating or not.

```
void DrawChildrenAlways (in ODShape invalidShape,
                         in ODCanvas biasCanvas);
```

`invalidShape`
A reference to a shape within which the embedded facets should draw, expressed in the frame coordinates of this facet.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

This facet's containing part calls this method to draw all embedded facets of this facet, whether they need updating or not. This method in turn calls the `Draw` method associated with this facet's embedded part.

SEE ALSO

The `ODFacet::DrawChildren` method (page 239).
The `ODPart::Draw` method (page 487).

DrawnIn

The `DrawnIn` method is called when this facet has been drawn in without an update event being generated.

```
void DrawnIn (in ODShape shape,
              in ODCanvas biasCanvas);
```

`shape` A reference to a shape object defining the area in this facet that was updated, expressed in frame coordinates.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

This facet's part calls this method when it has drawn asynchronously to the facet. This method in turn calls the `CanvasUpdated` method associated with the owner of this facet's canvas to notify it that its canvas has changed. The owning part decides when to process the update.

EXCEPTIONS

`kODErrFacetNotFound` The requested facet was not found.

SEE ALSO

The `ODPart::CanvasUpdated` method (page 469).

GetCanvas

The `GetCanvas` method returns a reference to the canvas associated with this facet.

```
ODCanvas GetCanvas ();
```

return value A reference to the canvas associated with this facet.

DISCUSSION

If this facet has no canvas of its own, this method searches recursively for the containing facet's canvas.

EXCEPTIONS

`kODErrCanvasNotFound`
Neither this facet nor any of its containing facets has a canvas.

SEE ALSO

The `ODFacet::ChangeCanvas` method (page 229).
The `ODFacet::HasCanvas` method (page 245).

GetContainingFacet

The `GetContainingFacet` method returns a reference to the containing facet of this facet.

```
ODFacet GetContainingFacet ();
```

return value A reference to the containing facet of this facet, or `kODNULL` if this facet has no containing facet.

GetFrame

The `GetFrame` method returns a reference to this facet's frame.

```
ODFrame GetFrame ();
```

return value A reference to this facet's frame.

DISCUSSION

This method increments the reference count of the returned frame. When you have finished using that frame, you should call its `Release` method.

GetHighlight

The `GetHighlight` method returns the highlight state for this facet.

```
ODHighlight GetHighlight ();
```

return value The highlight state for this facet. The return value is one of the following: `kODNoHighlight`, `kODFullHighlight`, or `kODDimHighlight`.

DISCUSSION

This facet's part uses the highlight state to draw its content consistently with the content highlighting of this facet's containing part.

SEE ALSO

The `ODHighlight` type (page 850).
The `ODFacet::ChangeHighlight` method (page 231).

GetPartInfo

The `GetPartInfo` method returns the part info data for this facet.

```
ODInfoType GetPartInfo ();
```

return value The part info data for this facet.

DISCUSSION

You should cast the return value to a pointer to your part's own representation of the data.

SEE ALSO

The `ODInfoType` type (page 853).
The `ODFacet::SetPartInfo` method (page 250).
The `ODFrame::GetPartInfo` method (page 315).

GetWindow

The `GetWindow` method returns a reference to the window this facet is displayed in.

```
ODWindow GetWindow ();
```

return value A reference to the window this facet is displayed in, or `kODNULL` if this facet does not actually appear in any window. For example, a printing facet does not appear in a window.

DISCUSSION

If this facet has no window of its own, this method searches recursively for the containing facet's window. Only the root facet of a window has a reference to the window; all its embedded facets then inherit this value.

This method increments the reference count of the returned window object. When you have finished using that window object, you should call its `Release` method.

SEE ALSO

The `ODFrame::AcquireWindow` method (page 299).

HasCanvas

The `HasCanvas` method returns a Boolean value that indicates whether this facet has its own canvas.

```
ODBoolean HasCanvas ();
```

return value `kODTrue` if this facet has its own canvas, otherwise `kODFalse`.

SEE ALSO

The `ODFacet::ChangeCanvas` method (page 229).
The `ODFacet::GetCanvas` method (page 241).

Invalidate

The `Invalidate` method marks the specified area in this facet as in need of updating.

```
void Invalidate (in ODShape invalidShape,
                 in ODCanvas biasCanvas);
```

`invalidShape`
A reference to the shape object defining the area in this facet that needs updating, expressed in frame coordinates.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

OpenDoc calls this method. This method transforms and clips the shape from the coordinate space of this facet to the coordinate space of its canvas. The shape is marked on the canvas (and all parent canvases) associated with this facet, ensuring that changes to this facet are reflected across all appropriate canvases.

EXCEPTIONS

`kODErrFacetNotFound`	The requested facet was not found.
`kODErrInvalidFacet`	The specified facet is not an embedded facet of this facet.

SEE ALSO

The `ODCanvas::Invalidate` method (page 74).
The `ODFacet::Validate` method (page 252).
The `ODFrame::Invalidate` method (page 317).

InvalidateActiveBorder

The `InvalidateActiveBorder` method marks the active frame border of this facet as in need of updating.

```
void InvalidateActiveBorder ();
```

DISCUSSION

OpenDoc calls this method when invalidating the active frame border if there is no explicit active frame border defined. When this facet's frame has the selection focus, OpenDoc automatically takes care of changing the facet's active border shape and redrawing the active frame border.

Under normal circumstances, there is no need to invalidate the active frame border for either the active part or its containing part. However, parts could call this method if they want to force the active frame border to be redrawn when they change the shape of an embedded part. To do so, the part must invalidate the active frame border by calling its facet's `InvalidateActiveBorder` method and redraw the active frame border by calling its facet's `DrawActiveBorder` method.

SEE ALSO

The `ODFacet::DrawActiveBorder` method (page 238).
The `ODFrame::InvalidateActiveBorder` method (page 318).

IsSelected

The `IsSelected` method returns a Boolean value that indicates whether this facet is selected within its containing part.

```
ODBoolean IsSelected ();
```

return value `kODTrue` if this facet is selected within its containing part, otherwise `kODFalse`.

DISCUSSION

A containing part calls this method when it believes this facet's frame has the selection focus. The selection criteria for facet selection is specific to the containing part that calls this method. OpenDoc uses the return value to determine whether to dispatch mouse events to this facet or to its containing facet.

SEE ALSO

The `ODFacet::SetSelected` method (page 250).

MoveBefore

The `MoveBefore` method repositions an embedded facet of this facet in front of a sibling facet.

```
void MoveBefore (in ODFacet child,
                 in ODFacet sibling);
```

`child`	A reference to an embedded facet to be repositioned.
`sibling`	A reference to a sibling facet that is to be directly behind the embedded facet. If the value is `kODNULL`, the embedded facet is repositioned in front of all its sibling facets.

DISCUSSION

After this method executes successfully, the sibling order associated with the embedded facets may or may not have changed. Any changes to the sibling order of the embedded facets are reflected in the facet iterator, if a facet iterator exists.

While you are using a facet iterator, you should not call this method to modify the list of embedded facets. You must postpone repositioning facets in the list of embedded facets until after you have deleted the iterator.

EXCEPTIONS

`kODErrInvalidFacet`	The child or sibling facet is not an embedded facet of this facet.

SEE ALSO

The `ODFacet::MoveBehind` method (page 248).

MoveBehind

The `MoveBehind` method repositions an embedded facet of this facet behind a sibling facet.

```
void MoveBehind (in ODFacet child,
                 in ODFacet sibling);
```

`child`	A reference to an embedded facet to be repositioned.

`sibling` A reference to a sibling facet that is to be directly in front of the embedded facet. If the value is `kODNULL`, the embedded facet is repositioned behind all its sibling facets.

DISCUSSION

After this method executes successfully, the sibling order associated with the embedded facets may or may not have changed. Any changes to the sibling order of the embedded facets are reflected in the facet iterator, if a facet iterator exists.

While you are using a facet iterator, you should not call this method to modify the list of embedded facets. You must postpone repositioning facets in the list of embedded facets until after you have deleted the iterator.

EXCEPTIONS

`kODErrInvalidFacet` The child or sibling facet is not an embedded facet of this facet.

SEE ALSO

The `ODFacet::MoveBefore` method (page 247).

RemoveFacet

The `RemoveFacet` method removes an embedded facet from this facet.

```
void RemoveFacet (in ODFacet facet);
```

`facet` A reference to an embedded facet to be removed.

DISCUSSION

This facet's containing part calls this method before removing one of its embedded frames, or optionally when scrolling an embedded frame out of view. This method in turn calls the `FacetRemoved` method associated with

this facet's part to notify the part that a facet has been removed from one of its frames. After this method executes successfully, the removed facet should not be used again; the caller should delete the removed facet.

EXCEPTIONS

`kODErrInvalidFacet`	The specified facet is not an embedded facet of this facet.

SEE ALSO

The `ODFrame::FacetRemoved` method (page 313).
The `ODPart::FacetRemoved` method (page 497).

SetPartInfo

The `SetPartInfo` method assigns part info data to this facet.

```
void SetPartInfo (in ODInfoType partInfo);
```

`partInfo` The data for this facet's part info.

SEE ALSO

The `ODInfoType` type (page 853).
The `ODFacet::GetPartInfo` method (page 243).
The `ODFrame::SetPartInfo` method (page 328).

SetSelected

The `SetSelected` method specifies whether this facet is currently being selected within its containing part.

```
void SetSelected (in ODBoolean isSelected);
```

`isSelected` `kODTrue` if this facet is currently being selected within its containing part, otherwise `kODFalse`.

DISCUSSION

The selection criteria for facet selection is specific to the containing part that calls this method.

SEE ALSO

The `ODFacet::IsSelected` method (page 247).

Update

The `Update` method updates this facet's canvas by drawing this facet and any of its embedded facets whose clip shape intersects the specified area of the canvas.

```
void Update (in ODShape invalidShape,
             in ODCanvas biasCanvas);
```

`invalidShape`
A reference to a shape object defining the area of the canvas that needs updating, expressed in frame coordinates.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

When an update event occurs that involves a facet of your part, OpenDoc calls its window's `Update` method, which in turn calls this method. This method in turn calls the `Draw` method associated with this facet's part.

SEE ALSO

The `ODPart::Draw` method (page 487).
The `ODWindow::Update` method (page 810).

Validate

The `Validate` method marks the specified area in this facet as no longer in need of updating.

```
void Validate (in ODShape validShape,
               in ODCanvas biasCanvas);
```

`validShape` A reference to a shape object defining the area in this facet that no longer needs updating, expressed in frame coordinates.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

OpenDoc calls this method. This method transforms and clips the shape from the coordinate space of this facet to the coordinate space of its canvas. It then subtracts the shape from any existing invalid area of the canvas.

SEE ALSO

The `ODCanvas::Validate` method (page 80).
The `ODFacet::Invalidate` method (page 245).
The `ODFrame::Validate` method (page 332).

ODFacetIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODFacetIterator` class provides access to all facets embedded in a facet.

Description

You use a facet iterator to apply an operation to a facet and all its embedded facets. For example, you might use a facet iterator to make sure that all facets within a facet being invalidated are also invalidated.

Your part creates a facet iterator object by calling a facet's `CreateFacetIterator` method (page 235), which returns a reference to a facet iterator object.

When you create a facet iterator, you specify the traversal type and sibling order. These two characteristics determine which facets are included in the iteration sequence and the order of the facets within the iteration sequence.

- If you specify top-down traversal, traverse the facet hierarchy top down, in depth-first order.
- If you specify bottom-up traversal, traverse the facet hierarchy bottom up, visiting the specified facet after visiting all its children. If the sibling order is front to back, the traversal starts with the frontmost facet at the lowest level in the hierarchy. If the sibling order is back to front, the traversal starts with the backmost facet at the lowest level in the hierarchy.
- If you specify children-only traversal, traverse only the children of the specified facet (not including the specified facet itself).

While you are using a facet iterator, you should not modify the list of embedded facets or call the facet's `MoveBefore` method (page 247) or `MoveBehind` method (page 248). You must postpone adding facets to or

removing facets from the list of embedded facets until after you have deleted the iterator.

For more information related to facet objects, see the `ODFacet` class description (page 215). For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODFacetIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and returns a reference to the first facet in the iteration sequence.
`Next`	Returns a reference to the next facet in the iteration sequence.
`SkipChildren`	Advances to the next sibling, skipping over the embedded facets of the current facet if the traversal type is topdown.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and returns a reference to the first facet in the iteration sequence, as indicated by the traversal type and sibling order that were set for this facet iterator.

```
ODFacet First ();
```

return value A reference to the first facet in the iteration sequence, or `kODNULL` if the traversal type is children-only and the root facet has no embedded facets.

DISCUSSION

The traversal type and sibling order is set by the facet's `CreateFacetIterator` method.

Your part must call this method before calling this facet iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

EXCEPTIONS

`kODErrIteratorOutOfSync` The list of embedded facets was modified while the iteration was in progress.

SEE ALSO

The `ODFacet::CreateFacetIterator` method (page 235).

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more facets remain in the iteration sequence. This method returns `kODTrue` if the preceding call to the `First` or

`Next` method found a facet. This method returns `kODFalse` when you have examined all the facets.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The list of embedded facets was modified while the iteration was in progress.

Next

The `Next` method returns a reference to the next facet in the iteration sequence.

```
ODFacet Next ();
```

return value A reference to the next facet in the iteration sequence, or `kODNULL` if you have reached the last facet.

DISCUSSION

If your part calls this method before calling this facet iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of embedded facets was modified while the iteration was in progress.

SkipChildren

The `SkipChildren` method advances to the next sibling, skipping over the embedded facets of the current facet if the traversal type is topdown.

```
void SkipChildren ();
```

DISCUSSION

If the traversal type for this facet iterator is not topdown, calling this method has no effect.

ODFocusModule

Superclasses `ODObject`

Subclasses none

An object of the `ODFocusModule` class is used to manage a particular type (or types) of focus or ownership of a shared resource.

Description

A focus is a designation of ownership of a given shared resource or feature, such as the keyboard or menu bar. A frame that owns an event-related focus receives events pertaining to that resource. A **focus module** manages a particular focus, maintaining the identity of the individual frame that owns the focus. The arbitrator uses at least one internal focus module to handle standard OpenDoc event types of a particular platform. Typically, you do not need to subclass `ODFocusModule` or even access the internal focus modules directly. However, you can define additional focus types, as needed, to handle other kinds of user events (such as input from new kinds of devices) by creating a new kind of focus module.

The `ODFocusModule` class is an abstract superclass that you can subclass to create a focus module. You can create a focus module object either from within a shell plug-in or from within one of your part's methods that are called during startup.

Before OpenDoc is able to recognize your new type of focus, you must register the associated focus module with the arbitrator. To register a focus module, you call the arbitrator's `RegisterFocus` method (page 50); to remove the focus module, call the arbitrator's `UnregisterFocus` method (page 58). For more information related to the arbitrator, see the `ODArbitrator` class description (page 43). For more information on creating custom focus types, see the chapter on extending OpenDoc in the *OpenDoc Programmer's Guide for the Mac OS.*

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODFocusModule`.

somInit

The `somInit` method initializes the instance variables in a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODFocusModule`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in this focus module object. The SOM library calls this method when this focus module is created. You must not do anything that might fail in this method. This limits you to operations like setting pointer variables to null, setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in this focus module's subclass-specific initialization method; see also the `InitFocusModule` method (page 266).

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODFocusModule`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for this focus module object, including any storage related to additional instance variables initialized in this focus module object. The SOM library calls this method when this focus module is deleted; this method must not fail.

Methods

This section presents summary descriptions of the `ODFocusModule` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Initializing

`InitFocusModule` Initializes this focus module object.

Transfer Focus

`TransferFocusOwnership`
Should transfer ownership of the specified focus from one frame to another frame.

`BeginRelinquishFocus`
Should return a Boolean value that indicates whether the current owner of the specified exclusive focus is willing to give up ownership of the focus.

`CommitRelinquishFocus`
Should signal to the part that owns the specified exclusive focus that it is about to lose ownership of it.

`AbortRelinquishFocus`
Should cancel the request for the frame to relinquish ownership of the specified exclusive focus.

Focus Ownership

`CreateOwnerIterator`
Should create a focus-owner iterator to give callers access to the frames that own the specified nonexclusive focus.

`AcquireFocusOwner` Should return a reference to the frame that owns the specified exclusive focus.

`SetFocusOwnership` Should record the specified frame as an owner of the specified focus.

`UnsetFocusOwnership` Should remove the specified frame as an owner of the specified focus.

Focus Testing

`IsFocusExclusive`	Should return a Boolean value that indicates whether the specified focus is exclusive.

AbortRelinquishFocus

The `AbortRelinquishFocus` method should cancel the request for the frame to relinquish ownership of the specified exclusive focus.

```
void AbortRelinquishFocus (in ODTypeToken focus,
                           in ODFrame requestingFrame);
```

`focus` A tokenized string representing the focus type that was to be relinquished, expressed as a 32-bit value.

`requestingFrame`
A reference to a frame that originally requested the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method, which in turns calls the `AbortRelinquishFocus` method of the part that owns the focus. This method should give those focus owners who have indicated willingness to relinquish the focus an opportunity to back out of changes initiated when OpenDoc first called the part's `BeginRelinquishFocus` method.

OVERRIDING

If you subclass `ODFocusModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODFocusModule::BeginRelinquishFocus` method (page 263).
The `ODFocusModule::CommitRelinquishFocus` method (page 264).
The `ODPart::AbortRelinquishFocus` method (page 461).
The `ODPart::BeginRelinquishFocus` method (page 467).
The `ODSession::Tokenize` method (page 598).

AcquireFocusOwner

The `AcquireFocusOwner` method should return a reference to the frame that owns the specified exclusive focus.

```
ODFrame AcquireFocusOwner (in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type whose owner is desired, expressed as a 32-bit value.
return value	A reference to the frame that owns the specified exclusive focus, or `kODNULL` if the focus is not owned by any frame.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method. A part can obtain a reference to the owner of a specified exclusive focus by calling the arbitrator's `AcquireFocusOwner` method, which in turn calls this method. If the focus is not registered, then the focus has no focus module and this method is never called.

Before returning the frame object, your override method should call the frame object's `Acquire` method. When the caller has finished using the returned frame object, it should call the frame object's `Release` method.

OVERRIDING

If you subclass `ODFocusModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODArbitrator::AcquireFocusOwner` method (page 262).
The `ODSession::Tokenize` method (page 598).

BeginRelinquishFocus

The `BeginRelinquishFocus` method should return a Boolean value that indicates whether the current owner of the specified exclusive focus is willing to give up ownership of the focus.

```
ODBoolean BeginRelinquishFocus (
                                in ODTypeToken focus,
                                in ODFrame requestingFrame);
```

`focus` A tokenized string representing the focus type to be relinquished, expressed as a 32-bit value.

`requestingFrame`
A reference to the frame requesting ownership of the focus.

return value `kODTrue` if the current owner of the specified exclusive focus is willing to give up ownership of the focus, otherwise `kODFalse`.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method, which in turns calls the `BeginRelinquishFocus` method of the part that owns the focus. If the part's `BeginRelinquishFocus` method returns `kODTrue` (the typical case), this method should return `kODTrue`; if the part's `BeginRelinquishFocus` method returns `kODFalse`, this method should return `kODFalse`.

OVERRIDING

If you subclass `ODFocusModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODFocusModule::AbortRelinquishFocus` method (page 261).
The `ODFocusModule::CommitRelinquishFocus` method (page 264).
The `ODPart::BeginRelinquishFocus` method (page 467).
The `ODSession::Tokenize` method (page 598).

CommitRelinquishFocus

The `CommitRelinquishFocus` method should signal to the part that owns the specified exclusive focus that it is about to lose ownership of it.

```
void CommitRelinquishFocus (in ODTypeToken focus,
                            in ODFrame requestingFrame);
```

`focus` A tokenized string representing the focus type to be relinquished, expressed as a 32-bit value.

`requestingFrame`
A reference to a frame that requested the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method, which in turns calls the `CommitRelinquishFocus` method of the part that owns the focus.

OVERRIDING

If you subclass `ODFocusModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODFocusModule::AbortRelinquishFocus` method (page 261).
The `ODFocusModule::BeginRelinquishFocus` method (page 263).
The `ODPart::CommitRelinquishFocus` method (page 472).
The `ODSession::Tokenize` method (page 598).

CreateOwnerIterator

The `CreateOwnerIterator` method should create a focus-owner iterator to give callers access to the frames that own the specified nonexclusive focus.

```
ODFocusOwnerIterator CreateOwnerIterator (
                                    in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type whose frames you want to enumerate, expressed as a 32-bit value.
return value	A reference to a new focus-owner iterator object, or `kODNULL` if the focus is exclusive.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method. This method should create and initialize an instance of a focus-owner iterator that can iterate over this focus module's focus owners, and return the iterator to the caller.

While you are using a focus-owner iterator, you should not modify the list of focus owners. You must postpone adding items to or removing items from the list of focus owners until after you have deleted the iterator.

OVERRIDING

If you subclass `ODFocusModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).
The `ODFocusOwnerIterator` class (page 272).

InitFocusModule

The `InitFocusModule` method initializes this focus module object.

```
void InitFocusModule (in ODSession session);
```

`session` A reference to the current session object.

DISCUSSION

This method is not called directly to initialize this focus module object, but is called by a subclass-specific initialization method. By convention, every subclass of `ODFocusModule` should have a separate initialization method (for example, the `InitMyFocusModule` method) that is called when an instance of that subclass is created. The override method may have additional parameters beyond those of the `InitFocusModule` method. The `InitMyFocusModule` method should call the inherited `InitFocusModule` method at the beginning of its implementation.

If you subclass `ODFocusModule`, your subclass-specific initialization method, rather than its `somInit` method, should handle any initialization code that can potentially fail. For example, your initialization method may attempt to allocate memory for your focus module.

OVERRIDING

If you subclass `ODFocusModule`, you should not override this method.

IsFocusExclusive

The `IsFocusExclusive` method should return a Boolean value that indicates whether the specified focus is exclusive.

```
ODBoolean IsFocusExclusive (in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type to be tested, expressed as a 32-bit value.
return value	`kODTrue` if the specified focus is exclusive, otherwise `kODFalse`.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the

tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

Foci may be exclusive or nonexclusive. All of the standard foci defined by OpenDoc are exclusive, meaning that only one frame at a time can own the focus. If you create a new kind of focus, you can make it nonexclusive, meaning that several frames could share ownership of it.

OVERRIDING

If you subclass `ODFocusModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

SetFocusOwnership

The `SetFocusOwnership` method should record the specified frame as an owner of the specified focus.

```
void SetFocusOwnership (in ODTypeToken focus,
                        in ODFrame frame);
```

`focus`	A tokenized string representing the focus whose owner is to be assigned, expressed as a 32-bit value.
`frame`	A reference to a frame that is to own the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`,

kODMouseFocus, kODScrollingFocus, or kODSelectionFocus) or the tokenized form of a part-specific focus type. You can call the session object's Tokenize method to obtain a token corresponding to one of these constants.

OpenDoc calls this method. This method should record, in this focus module's internal structures, the new focus ownership.

OVERRIDING

If you subclass ODFocusModule, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The ODFocusType type (page 859).
The ODTypeToken type (page 847).
The ODSession::Tokenize method (page 598).

TransferFocusOwnership

The TransferFocusOwnership method should transfer ownership of the specified focus from one frame to another frame.

```
void TransferFocusOwnership (in ODTypeToken focus,
                             in ODFrame transferringFrame,
                             in ODFrame newOwner);
```

`focus` A tokenized string representing the focus type to be transferred, expressed as a 32-bit value.

`transferringFrame`
A reference to a frame relinquishing ownership of the focus; it does not have to be the current owner of the focus.

`newOwner` A reference to a frame obtaining ownership of the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method. This method should record, in this focus module's internal structures, the transfer of focus ownership.

OVERRIDING

If you subclass `ODFocusModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

UnsetFocusOwnership

The `UnsetFocusOwnership` method should remove the specified frame as an owner of the specified focus.

```
void UnsetFocusOwnership (in ODTypeToken focus,
                          in ODFrame frame);
```

`focus` A tokenized string representing the focus whose owner is to be removed, expressed as a 32-bit value.

`frame` A reference to a current owner of the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method. This method should record, in this focus module's internal structures, the loss of focus ownership. This method can be called for both exclusive and nonexclusive foci.

OVERRIDING

If you subclass `ODFocusModule`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

ODFocusOwnerIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODFocusOwnerIterator` class provides access to all owners of a nonexclusive focus.

Description

Foci may be exclusive or nonexclusive. All of the standard foci defined by OpenDoc are **exclusive**, meaning that only one frame at a time can own a focus. If you create a new kind of focus, you can make it **nonexclusive**, meaning that several frames could share ownership of it.

You must create a focus-owner iterator if you create a focus module for a nonexclusive focus. The focus module keeps a list of all the individual frames that own the nonexclusive focus. A focus-owner iterator contains a reference to the focus module that manages the nonexclusive focus and provides access to all of the owner frames.

You use a focus-owner iterator to apply an operation to all owners of a nonexclusive focus. For example, a part might use a focus-owner iterator to notify all frames that own a video input focus to synchronize themselves so that the video is displayed in all of the frames simultaneously.

The `ODFocusOwnerIterator` class is an abstract superclass that you can subclass to create a focus-owner iterator. Your part creates a focus-owner iterator object by calling the arbitrator's `CreateOwnerIterator` method (page 47). OpenDoc in turn calls the appropriate focus module's `CreateOwnerIterator` method (page 265), which returns a reference to a focus-owner iterator object.

While you are using a focus-owner iterator, you should not modify the list of focus owners. You must postpone adding frames to or removing frames from the list of focus owners until after you have deleted the iterator.

For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODFocusOwnerIterator`.

somInit

The `somInit` method initializes the instance variables in a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODFocusOwnerIterator`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in this focus-owner iterator object. The SOM library calls this method when this focus-owner iterator is created. You must not do anything that might fail in this method. This limits you to operations like setting pointer variables to null, setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in this focus-owner iterator's subclass-specific initialization method; see also the `InitFocusOwnerIterator` method (page 275).

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODFocusOwnerIterator`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for this focus-owner iterator object, including any storage related to additional instance variables initialized in this focus-owner iterator object. The SOM library calls

this method when this focus-owner iterator is deleted; this method must not fail.

Methods

This section presents summary descriptions of the `ODFocusOwnerIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Initializing

`InitFocusOwnerIterator` Initializes this focus-owner iterator object.

Accessing

`First` Should begin the iteration and return a reference to the first frame in the iteration sequence.

`Next` Should return a reference to the next frame in the iteration sequence.

Iterator Testing

`IsNotComplete` Should return a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method should begin the iteration and return a reference to the first frame in the iteration sequence.

```
ODFrame First ();
```

return value A reference to the first frame in the iteration sequence, or `kODNULL` if the focus has no owners.

DISCUSSION

Your part calls this method before calling this focus-owner iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

Your override of this method should not increment the reference count of the returned frame object.

OVERRIDING

If you subclass `ODFocusOwnerIterator`, you must override this method. Your override must not call its inherited method; that is, your override must implement this method's functionality completely.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of focus owners was modified while the iteration was in progress.

InitFocusOwnerIterator

The `InitFocusOwnerIterator` method initializes this focus-owner iterator object.

```
void InitFocusOwnerIterator (in ODTypeToken focus,
                             in ODFocusModule focusModule);
```

`focus`
: A tokenized string representing the nonexclusive focus type of the owners returned by this iterator, expressed as a 32-bit value.

`focusModule`
: A reference to a focus module that lists the owners of the specified focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

This method is not called directly to initialize this focus-owner iterator object, but is called by a subclass-specific initialization method. By convention, every subclass of `ODFocusOwnerIterator` should have a separate initialization method (for example, the `InitMyFocusOwnerIterator` method) that is called when an instance of that subclass is created. The override method may have additional parameters beyond those of the `InitFocusOwnerIterator` method. The `InitMyFocusOwnerIterator` method should call the inherited `InitFocusOwnerIterator` method at the beginning of its implementation.

If you subclass `ODFocusOwnerIterator`, your subclass-specific initialization method, rather than its `somInit` method, should handle any initialization code that can potentially fail. For example, your initialization method may attempt to allocate memory for your focus-owner iterator.

OVERRIDING

If you subclass `ODFocusOwnerIterator`, you must not override this method.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

IsNotComplete

The `IsNotComplete` method should return a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more focus owners remain in the iteration sequence. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found a focus owner. This method returns `kODFalse` when you have examined all the focus owners.

OVERRIDING

If you subclass `ODFocusOwnerIterator`, you must override this method. Your override must not call its inherited method; that is, your override must implement this method's functionality completely.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The list of focus owners was modified while the iteration was in progress.

Next

The `Next` method should return a reference to the next frame in the iteration sequence.

```
ODFrame Next ();
```

return value A reference to the next frame in the iteration sequence, or `kODNULL` if you have reached the last frame.

DISCUSSION

Your part calls this method. If your part calls this method before calling this focus-owner iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

Your override of this method should not increment the reference count of the returned frame object.

OVERRIDING

If you subclass `ODFocusOwnerIterator`, you must override this method. Your override must not call its inherited method; that is, your override must implement this method's functionality completely.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of focus owners was modified while the iteration was in progress.

ODFocusSet

Superclasses `ODObject`
Subclasses none

An object of the `ODFocusSet` class provides grouping of foci for activation.

Description

A focus is a designation of ownership of a given shared resource or feature, such as the keyboard or menu bar. A frame that owns a focus receives events pertaining to that resource. Foci may be manipulated singly or in groups called **focus sets**. A focus set is a list of foci that can be obtained and released as a group. For example, if a frame wants to request ownership of keyboard and menu foci together, it can create a focus set that includes these two focus types. If a frame requests a focus set and one or more of the foci included in the focus set is not available, ownership of the entire focus set is denied.

Your part creates a focus set object by calling the arbitrator's `CreateFocusSet` method (page 46), which returns a reference to a focus set object. Frames obtain ownership of focus sets by calling the arbitrator's `RequestFocusSet` method (page 54). For more information about foci and focus sets, see the `ODFocusSetIterator` class description (page 284) and the chapter on user events in the *OpenDoc Programmer's Guide for the Mac OS.*

Methods

This section presents summary descriptions of the `ODFocusSet` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Iterating

`CreateIterator` Creates a focus-set iterator object for this focus set.

Adding and Removing Foci

`Add`	Adds the specified focus to this focus set.
`Remove`	Removes the specified focus from this focus set.

Testing

`Contains`	Returns a Boolean value that indicates whether the specified focus is a member of this focus set.

Add

The `Add` method adds the specified focus to this focus set.

```
void Add (in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type to be added to this focus set, expressed as a 32-bit value.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

The specified focus is not added if it already exists in this focus set.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to expand the focus set.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

Contains

The `Contains` method returns a Boolean value that indicates whether the specified focus is a member of this focus set.

```
ODBoolean Contains (in ODTypeToken focus);
```

`focus`	A tokenized string representing the focus type to be tested for membership, expressed as a 32-bit value.
return value	`kODTrue` if the specified focus is a member of this focus set, otherwise `kODFalse`.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

CreateIterator

The `CreateIterator` method creates a focus-set iterator object for this focus set.

```
ODFocusSetIterator CreateIterator ();
```

return value A reference to a new focus-set iterator object.

DISCUSSION

While you are using a focus-set iterator, you should not modify the focus set. You must postpone adding items to or removing items from the focus set until after you have deleted the iterator.

SEE ALSO

The `ODFocusSetIterator` class (page 284).

Remove

The `Remove` method removes the specified focus from this focus set.

```
void Remove (in ODTypeToken focus);
```

`focus` A tokenized string representing the focus type to be removed from the focus set, expressed as a 32-bit value.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).

ODFocusSetIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODFocusSetIterator` class provides access to all foci in a focus set.

Description

You use a focus-set iterator to apply an operation to all foci in a focus set. For example, you might store a focus set with all the foci owned by a given frame, and then use a focus-set iterator to search for a particular focus in the focus set.

Your part creates a focus-set iterator object by calling a focus set's `CreateIterator` method (page 282), which returns a reference to a focus-set iterator object.

While you are using a focus-set iterator, you should not modify or delete the focus set. You must postpone adding foci to or removing foci from the focus set until after you have deleted the iterator.

For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODFocusSetIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and returns the first focus in the focus set.
`Next`	Returns a reference to the next focus in the focus set.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and returns the first focus in the focus set.

```
ODTypeToken First ();
```

return value A tokenized string representing the first focus in the focus set, or `kODNullFocus` for an empty focus set.

DISCUSSION

Your part must call this method before calling this focus-set iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

EXCEPTIONS

`kODErrIteratorOutOfSync`
The focus set was modified while the iteration was in progress.

SEE ALSO

The `ODTypeToken` type (page 847).

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more foci remain in the focus set. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found a focus. This method returns `kODFalse` when you have examined all the foci. If the focus set is empty, this method always returns `kODFalse`.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The focus set was modified while the iteration was in progress.

Next

The `Next` method returns a reference to the next focus in the focus set.

```
ODTypeToken Next ();
```

return value A tokenized string representing the next focus in the focus set, or `kODNullFocus` if you have reached the end of the focus set.

DISCUSSION

If your part calls this method before calling this focus-set iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The focus set was modified while the iteration was in progress.

SEE ALSO

The `ODTypeToken` type (page 847).

ODFrame

Superclasses `ODPersistentObject` → `ODRefCntObject` → `ODObject`
Subclasses none

An object of the `ODFrame` class describes the display area of an embedded part in the content of its containing part.

Description

A frame object represents an area of content of a part; it marks the geometric boundary between an embedded part and its containing part. There are several possible configurations and states for frames:

- A part's **display frame** is a frame within which the part's content is drawn. Each display frame represents a particular view of a part's content.
- A part's **embedded frame** is a frame within which one of the part's embedded parts is displayed.
- A **containing frame** is a frame in which one or more frames is embedded. Each embedded frame has one containing frame; each containing frame has one or more embedded frames.
- The **root frame** is the frame in which the root part of a window is displayed. The root frame shape is the same as the content area of the window.
- The **active frame** is the frame that has the selection focus. Editing takes place in the active frame; that frame displays the selection or insertion point.
- An **overlaid frame** is an embedded frame that floats above a containing part's content (including other embedded frames). Overlaid frames do not need to negotiate for space except as required by the constraints of the containing part size.
- A **subframe** is a frame that is both an embedded frame in, and a display frame of, a part. A part can create an embedded frame, make it a subframe of its own display frame, and then display itself in that subframe. When a

subframe has the selection focus, the active frame border is displayed around its containing frame.

- A **bundled frame** is a frame whose content does not respond to geometry-based user events. A mouse click within a bundled frame selects the frame's part, but does not activate it.
- A **nonpersistent frame** is a frame that exists as an in-memory object only. A nonpersistent frame does not have a storage unit and is not stored persistently.

Your part creates a new frame object for its embedded parts by calling its draft's `CreateFrame` method (page 162). Your part accesses a previously stored frame object by calling its draft's `AcquireFrame` method (page 151). These methods return a reference to a frame object.

Frame Geometry

There are several things that define the geometry of a frame.

- The **frame shape** defines the area that the containing part propagates to an embedded part's display frame. The frame's containing part controls the frame shape.
- The **used shape** defines the area of an embedded part's frame that has actual content to display, that is, the part of the frame that the containing part should not draw over. However, the containing part is free to wrap content to the contour of the used shape. The embedded part controls the used shape.
- The **internal transform** describes how the part's content is positioned, scaled, or otherwise transformed within the frame. It represents the mapping from the coordinate space of the frame's content to the coordinate space of the frame.
- The **content extent** describes how much offset is used to calculate the bias transform, that is, the vertical extent of the content area of a part in a frame (in essence, the height of a part's page). For more information related to bias transforms, see the `ODCanvas` class description (page 61).

A frame must always possess a valid frame shape. The used shape does not need to be set; if it is not, the used shape is the same as the frame shape. For more information related to shape objects, see the `ODShape` class description (page 606).

A frame must always possess a valid internal transform which, if not explicitly set, is the identity transform. For more information related to transform objects, see the `ODTransform` class description (page 736).

The `ODFrame` class includes several methods that specify geometry (shape and transform) objects. Because these objects necessarily assume a coordinate system, the `ODFrame` methods include a parameter, `biasCanvas`, that allows you to specify a canvas to whose coordinate space the geometry is biased. The bias canvas uses a bias transform to convert from the coordinate system used for drawing on the canvas to the coordinate system (platform-normal coordinates) used by the current graphics system. When the bias canvas is specified, it is automatically applied to the returned object.

Each part in an OpenDoc document controls the position, size, and shape of the frames embedded within it. At the same time, embedded parts may want to change the size, shape, or number of frames they are displayed in. Through a process called **frame negotiation**, an embedded part and its containing part agree on the frame or frames the embedded part displays in. Either part can initiate the negotiation, although the containing part has unilateral control over the outcome.

Frame Hierarchy

A frame always maintains a reference to its part. The frame ensures that it is registered with its part on that part's internal list of display frames when the frame is created (by calling its part's `DisplayFrameAdded` method (page 478)) and that it is removed from that part's internal list of display frames when the frame is deleted (by calling its part's `DisplayFrameRemoved` method (page 481)).

A frame maintains a reference to its containing frame. The value of the reference is null if the specified frame is the root frame of a window. The reference value does not usually change, unless the frame was created before an embedding location was selected. A frame does not hold direct references to its embedded frames. Only the frame's part knows which frames are embedded in the frame.

A frame maintains a list of its facets. Facets hold nonpersistent information about the layout of parts, or describe the location of a frame on a particular canvas for display and event dispatching. There may be more than one facet per frame. The list may be empty if the specified frame is currently scrolled out

of view or otherwise not visible. For more information related to facet objects, see the `ODFacet` class description (page 215).

Display Information

There are two characteristics that describe how a part is displayed within a frame.

- The **view type** describes the basic visual representation of a part. Parts must support the standard set of view types (large icon, small icon, thumbnail, or frame view). The view type should be set only by the frame's part.
- The **presentation** of a frame describes, for parts whose view type is framed, a particular style of display for a part's content within the frame—for example, table, bar chart or pie chart, text or outline. Presentations are part-defined; your part editor determines what types of presentations your part is capable of, and defines a presentation designation for each. A containing part may request a particular presentation but the embedded part need not honor that request. The presentation should be set only by the frame's part.

A frame's part might store user-defined data in the frame's part info. The part alone interprets or manipulates the data, but the part info data is stored with the frame and read in only when it is necessary for frame manipulation.

Propagating Events

If one of your part's embedded frames propagates unhandled events to your part, your part has the opportunity to handle those events. In that case, your part's event handler needs to determine whether a particular frame is one of its display frames or an embedded frame.

Frame Groups

A **frame group** is a set of display frames that a part designates as related, for purposes such as flowing content from one frame to another. Each frame group has its own **group ID**; frames within a frame group have a **sequence number** that is used to define the position of a frame within its frame group. The group ID and sequence number should be set only by the frame's containing part.

Flags and Link Status

When a part creates a frame, the part sets flags, defined for the lifetime of the frame, that determine whether the frame is a root frame, a subframe, or an overlaid frame; once set, these flags cannot be changed.

A containing part must set the link status for any frame it embeds, even if the embedded part does not support links to its content. The containing part specifies a link status value based solely on the embedded frame's inclusion in links maintained by the containing part. A frame considers the link status of its containing frame when setting its own link status.

Methods

This section presents summary descriptions of the `ODFrame` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Frame Hierarchy

`AcquireContainingFrame`
: Returns a reference to a containing frame of this frame.

`SetContainingFrame`
: Assigns the specified frame as the containing frame of this frame.

`AcquirePart`
: Returns a reference to a part displayed in this frame.

`ChangePart`
: Assigns a new part to this frame.

`AcquireWindow`
: Returns a reference to the window this frame is displayed in.

`SetWindow`
: Assigns a window to this frame.

`IsSubframe`
: Returns a Boolean value that indicates whether this frame is a subframe of its containing frame.

`SetSubframe`
: Specifies whether this frame is currently a subframe of its containing frame.

Frame Geometry

`CreateShape`
: Creates a shape object.

`AcquireFrameShape`
: Returns a reference to the frame shape of this frame.

`ChangeFrameShape`
: Changes the frame shape of this frame.

`AcquireUsedShape`	Returns a reference to the used shape of this frame.
`ChangeUsedShape`	Assigns a used shape to this frame.
`CreateTransform`	Creates a transform object.
`AcquireInternalTransform`	Returns a reference to the internal transform of this frame.
`ChangeInternalTransform`	Changes the internal transform of this frame.
`RequestFrameShape`	Requests a new frame shape for this frame.

Frame Characteristics

`IsFrozen`	Returns a Boolean value that indicates whether this frame is bundled.
`SetFrozen`	Specifies whether this frame is currently bundled.
`IsInLimbo`	Returns a Boolean value that indicates whether this frame is removed from a part's content model.
`SetInLimbo`	Specifies whether this frame is to be removed from a part's content model.
`IsOverlaid`	Returns a Boolean value that indicates whether this frame is an overlaid frame.
`IsRoot`	Returns a Boolean value that indicates whether this frame is the root frame.

Closing and Removing Frames

`Close`	Prepares this frame to be removed from memory, but does not affect persistent storage.
`Remove`	Prepares this frame to be removed both from memory and persistent storage.

Frame Groups

`GetFrameGroup`	Returns the group ID of this frame.
`SetFrameGroup`	Assigns a group ID to this frame.
`GetSequenceNumber`	Returns the sequence number of this frame in its frame group.

`ChangeSequenceNumber`	Assigns a sequence number to this frame in its frame group.

Imaging

`GetViewType`	Returns this frame's view type.
`SetViewType`	Assigns the specified view type to this frame.
`ChangeViewType`	Changes the specified view type of this frame.
`GetPresentation`	Returns this frame's presentation.
`SetPresentation`	Assigns the specified presentation to this frame.
`ChangePresentation`	Changes the presentation of this frame.
`DrawActiveBorder`	Updates the active frame border of this frame.
`Invalidate`	Marks the specified area in this frame as in need of updating.
`InvalidateActiveBorder`	Marks the active frame border of this frame as in need of updating.
`Validate`	Marks the specified area in this frame as no longer in need of updating.

Linking

`EditInLink`	Returns a Boolean value that indicates whether the part maintaining a link destination (that includes this frame) can be found.
`GetLinkStatus`	Returns the link status of this frame.
`ChangeLinkStatus`	Changes the link status of this frame.
`ContentUpdated`	Notifies this frame's containing part that this frame's part has updated its content, so the containing part can update any link sources that it maintains.

Event Propagation

`DoesPropagateEvents`	Returns a Boolean value that indicates whether this frame propagates unhandled events to its containing part.
`SetPropagateEvents`	Specifies whether this frame should propagate unhandled events to its containing frame.

Facet Manipulation

`CreateFacetIterator`	Creates a frame-facet iterator object for the facets of this frame.
`FacetAdded`	Adds the facet to this frame's list of facets and notifies its part of the new facet.
`FacetRemoved`	Removes the facet from this frame's list of facets and notifies its part of the deletion.

Part Info

`GetPartInfo`	Returns the part info data for this frame.
`SetPartInfo`	Assigns part info data to this frame.

Content Extent

`GetContentExtent`	Gets the content extent of this frame.
`ChangeContentExtent`	Changes the content extent of this frame.

Drag and Drop

`IsDragging`	Returns a Boolean value that indicates whether this frame is currently being dragged.
`SetDragging`	Specifies whether this frame is currently being dragged.
`IsDroppable`	Returns a Boolean value that indicates whether this frame's part accepts drag-and-drop events in this frame.
`SetDroppable`	Specifies whether this frame's part accepts drag-and-drop events in this frame.

AcquireContainingFrame

The `AcquireContainingFrame` method returns a reference to a containing frame of this frame.

```
ODFrame AcquireContainingFrame ();
```

return value A reference to a containing frame of this frame, or `kODNULL` if this frame is the root frame.

DISCUSSION

This method increments the reference count of the returned frame. When you have finished using that frame, you should call its `Release` method.

SEE ALSO

The `ODFrame::SetContainingFrame` method (page 324).

AcquireFrameShape

The `AcquireFrameShape` method returns a reference to the frame shape of this frame.

```
ODShape AcquireFrameShape (in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to the frame shape, expressed in frame coordinates.

DISCUSSION

The caller must not modify the frame shape; only this frame's containing part can modify the frame shape.

This method increments the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

SEE ALSO

The `ODFrame::ChangeFrameShape` method (page 300).
The `ODFrame::RequestFrameShape` method (page 323).

AcquireInternalTransform

The `AcquireInternalTransform` method returns a reference to the internal transform of this frame.

```
ODTransform AcquireInternalTransform (
                                    in ODCanvas biasCanvas);
```

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

return value A reference to the internal transform of this frame.

DISCUSSION

The caller must not modify the internal transform; only this frame's part can modify the internal transform.

This method increments the reference count of the returned transform object. When you have finished using that transform object, you should call its `Release` method.

SEE ALSO

The `ODFrame::ChangeInternalTransform` method (page 301).

AcquirePart

The `AcquirePart` method returns a reference to a part displayed in this frame.

```
ODPart AcquirePart ();
```

return value A reference to a part displayed in this frame.

DISCUSSION

This method increments the reference count of the returned part. When you have finished using the part, you should call its `Release` method.

SEE ALSO

The `ODFrame::ChangePart` method (page 303).

AcquireUsedShape

The `AcquireUsedShape` method returns a reference to the used shape of this frame.

```
ODShape AcquireUsedShape (in ODCanvas biasCanvas);
```

`biasCanvas`	`kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.
return value	A reference to the used shape of this frame, expressed in frame coordinates.

DISCUSSION

The caller must not modify the used shape; only this frame's part can modify the used shape.

This method increments the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

SEE ALSO

The `ODFrame::ChangeUsedShape` method (page 305).

AcquireWindow

The `AcquireWindow` method returns a reference to the window this frame is displayed in.

```
ODWindow AcquireWindow ();
```

return value A reference to the window this frame is displayed in, or `kODNULL` if this frame does not actually appear in any window. For example, a printing frame does not appear in a window.

DISCUSSION

Only the root frame of a window has a reference to the window; all its embedded frames then inherit this value.

This method increments the reference count of the returned window object. When you have finished using that window object, you should call its `Release` method.

SEE ALSO

The `ODFacet::GetWindow` method (page 244).
The `ODFrame::SetWindow` method (page 331).

ChangeContentExtent

The `ChangeContentExtent` method changes the content extent of this frame.

```
void ChangeContentExtent (in ODPoint contentExtent);
```

`contentExtent`
The content extent to assign to this frame.

DISCUSSION

Content extent is, in essence, the page height of the part displayed in this frame. This method causes the bias transform associated with this frame's content to be recomputed.

SEE ALSO

The `ODPoint` type (page 855).
The `ODFrame::GetContentExtent` method (page 313).

ChangeFrameShape

The `ChangeFrameShape` method changes the frame shape of this frame.

```
void ChangeFrameShape (in ODShape shape,
                       in ODCanvas biasCanvas);
```

`shape` A reference to a frame shape to assign to this frame, expressed in frame coordinates.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

Your part calls its embedded frame's `ChangeFrameShape` method when it changes the shape of the embedded frame. This method in turn calls the `FrameShapeChanged` method of the embedded frame's part to notify the part that its frame shape has changed.

If the used shape is the same as the frame shape, this method also calls the `UsedShapeChanged` method of the embedded frame's part to notify the part that its used shape has changed.

EXCEPTIONS

`kODErrIllegalNullShapeInput`
The `shape` parameter is null.

SEE ALSO

The `ODFrame::AcquireFrameShape` method (page 296).
The `ODFrame::RequestFrameShape` method (page 323).
The `ODPart::FrameShapeChanged` method (page 501).
The `ODPart::UsedShapeChanged` method (page 529).

ChangeInternalTransform

The `ChangeInternalTransform` method changes the internal transform of this frame.

```
void ChangeInternalTransform (in ODTransform transform,
                              in ODCanvas biasCanvas);
```

`transform` A reference to an internal transform to associate with this frame.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

Your part calls its display frame's `ChangeInternalTransform` method to change the position of its content in the frame.

EXCEPTIONS

`kODErrIllegalNullTransformInput`
The `transform` parameter is null.

SEE ALSO

The `ODFrame::AcquireInternalTransform` method (page 297).

ChangeLinkStatus

The `ChangeLinkStatus` method changes the link status of this frame.

```
void ChangeLinkStatus (in ODLinkStatus status);
```

`status` The link status to assign to this frame. The value of `status` must be one of the following: `kODInLinkDestination`, `kODInLinkSource`, or `kODNotInLink`.

DISCUSSION

The value `kODInLinkDestination` for the `status` parameter indicates that the frame is embedded in the destination of a link; the content of this frame is thus supplied by a link. The value `kODInLinkSource` indicates that the frame is embedded in content that is the source of one or more links, but not in content that is the destination of a link. The value `kODNotInLink` indicates that the frame is not embedded in any linked content, source or destination.

If your part supports linking or embedding, your part calls this method for each embedded frame that is involved in a link when a link is created, broken, or moved. If your part supports linking only, your part calls this method for each frame not in a link. This method in turn calls the `LinkStatusChanged` method associated with this frame's part to notify the part that its link status has changed. In turn, the embedded part's `LinkStatusChanged` method gives that embedded part a chance to call the `ChangeLinkStatus` method for other embedded frames.

EXCEPTIONS

`kODErrInvalidLinkStatus` The specified link status is invalid.

SEE ALSO

The `ODLinkStatus` type (page 852).
The `ODFrame::GetLinkStatus` method (page 314).
The `ODPart::LinkStatusChanged` method (page 511).

ChangePart

The `ChangePart` method assigns a new part to this frame.

```
void ChangePart (in ODPart part);
```

`part` A reference to a part to associate with this frame.

DISCUSSION

Your part calls this method when it wants to keep one embedded frame and swap parts in and out of it, as is typical for browser-type parts. This method in turn calls the `FacetRemoved` method associated with this frame's part to remove all facets of this frame from its previous part. This method can also call the `FacetAdded` method associated with this frame's new part to add all facets associated with the previous part.

Unless this frame was removed or closed, you can always assume that this frame previously had a part.

EXCEPTIONS

`kODErrIllegalNullPartInput`
The `part` parameter is null.

SEE ALSO

The `ODPart::FacetAdded` method (page 496).
The `ODPart::FacetRemoved` method (page 497).

ChangePresentation

The `ChangePresentation` method changes the presentation of this frame.

```
void ChangePresentation (in ODTypeToken presentation);
```

`presentation`
A tokenized string representing the presentation to assign to this frame, expressed as a 32-bit value.

DISCUSSION

Your part calls its embedded frame's `ChangePresentation` method to request that it use a new presentation. This method in turn calls the `PresentationChanged` method of this frame's part to notify the part that its presentation has changed. If the embedded part does not support the requested presentation, it can pick a presentation that it can support and then call its frame's `SetPresentation` method to update the presentation in the frame.

EXCEPTIONS

`kODErrIllegalNullTokenInput`
The `presentation` parameter is null.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODFrame::GetPresentation` method (page 316).
The `ODFrame::SetPresentation` method (page 328).
The `ODPart::PresentationChanged` method (page 515).

ChangeSequenceNumber

The `ChangeSequenceNumber` method assigns a sequence number to this frame in its frame group.

```
void ChangeSequenceNumber (in ODULong sequenceNumber);
```

`sequenceNumber`
The sequence number to assign to this frame, expressed as an unsigned 32-bit value.

DISCUSSION

Your part calls its embedded frame's `ChangeSequenceNumber` method to reorder the sequence of its display frames in its frame group.

SEE ALSO

The `ODFrame::GetSequenceNumber` method (page 316).

ChangeUsedShape

The `ChangeUsedShape` method assigns a used shape to this frame.

```
void ChangeUsedShape (in ODShape shape,
                      in ODCanvas biasCanvas);
```

`shape` A reference to a used shape to assign to this frame, expressed in frame coordinates, or `kODNULL` if the used shape is the same as the frame shape.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

Your part assigns a used shape to its display frame by calling the frame's `ChangeUsedShape` method. This method in turn calls the `UsedShapeChanged` method of the display frame's containing part to notify the containing part that its used shape has changed.

SEE ALSO

The `ODFrame::AcquireUsedShape` method (page 298).
The `ODPart::UsedShapeChanged` method (page 529).

ChangeViewType

The `ChangeViewType` method changes the specified view type of this frame.

```
void ChangeViewType (in ODTypeToken viewType);
```

`viewType` A tokenized string representing the view type to assign to this frame.

DISCUSSION

The `viewType` parameter must be the tokenized form of one of the view-type constants (`kODViewAsFrame`, `kODViewAsSmallIcon`, `kODViewAsLargeIcon`, or `kODViewAsThumbnail`). You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

Your part calls its embedded frame's `ChangeViewType` method to request that it use a new view type. This method in turn calls the `ViewTypeChanged` method of this frame's part to notify the part that its view type has changed. If the embedded part does not support the requested view type, it can pick a view type that it can support and then call its frame's `SetViewType` method to update the view type in the frame.

EXCEPTIONS

`kODErrIllegalNullTokenInput`
The `viewType` parameter is null.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODFrame::GetViewType` method (page 316).
The `ODFrame::SetViewType` method (page 330).
The `ODPart::ViewTypeChanged` method (page 530).
The `ODSession::Tokenize` method (page 598).

Close

The `Close` method prepares this frame to be removed from memory, but does not affect persistent storage.

```
void Close ();
```

DISCUSSION

Your part calls its embedded frame's `Close` method when a document is being closed after its final save. This method in turn calls the `DisplayFrameClosed` method associated with this frame's part to notify the part that it is being closed.

During execution of this method, this frame releases its references to the associated part and its containing frame, and this frame should not be used again.

Depending on whether you want to affect persistent storage, your part calls either this method or its frame's `Remove` method.

SEE ALSO

The `ODFrame::Remove` method (page 322).
The `ODPart::DisplayFrameClosed` method (page 479).

ContentUpdated

The `ContentUpdated` method notifies this frame's containing part that this frame's part has updated its content, so the containing part can update any link sources that it maintains.

```
void ContentUpdated (in ODUpdateID change);
```

`change` The update ID associated with this frame.

DISCUSSION

Your part calls its display frame's `ContentUpdated` method when your part's content changes. Your part should avoid calling this method at every content change; it can probably call this method after a reasonable pause and when the part loses the selection focus. This method may be called multiple times. This method in turn calls the `EmbeddedFrameUpdated` method for all containing parts in the frame hierarchy through the root part of the window, so that any affected link sources can be updated.

SEE ALSO

The `ODUpdateID` type (page 887).
The `ODPart::EmbeddedFrameUpdated` method (page 494).

CreateFacetIterator

The `CreateFacetIterator` method creates a frame-facet iterator object for the facets of this frame.

```
ODFrameFacetIterator CreateFacetIterator ();
```

return value A reference to a new frame-facet iterator object.

DISCUSSION

Your part calls this method if it needs to apply an operation to all facets of one of your display frames, such as drawing and changing their shapes. It is your responsibility to delete the iterator when it is no longer needed.

While you are using a frame-facet iterator, you should not modify the list of facets for the frame. You must postpone adding items to or removing items from the list of facets for the frame until after you have deleted the iterator.

SEE ALSO

The `ODFacet::CreateFacetIterator` method (page 235).
The `ODFrameFacetIterator` class (page 333).

CreateShape

The `CreateShape` method creates a shape object.

```
ODShape CreateShape ();
```

return value A reference to a new shape object.

DISCUSSION

Your part calls this method to create a shape object for any purpose.

This method initializes the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

SEE ALSO

The `ODFacet::CreateShape` method (page 236).
The `ODShape` class (page 606).

CreateTransform

The `CreateTransform` method creates a transform object.

```
ODTransform CreateTransform ();
```

return value A reference to a new transform object.

DISCUSSION

Your part calls this method to create a transform object for any purpose.

This method initializes the reference count of the returned transform object. When you have finished using that transform object, you should call its `Release` method.

SEE ALSO

The `ODFacet::CreateTransform` method (page 237).
The `ODTransform` class (page 736).

DoesPropagateEvents

The `DoesPropagateEvents` method returns a Boolean value that indicates whether this frame propagates unhandled events to its containing part.

```
ODBoolean DoesPropagateEvents ();
```

return value `kODTrue` if this frame propagates unhandled events to its containing part, otherwise `kODFalse`.

DISCUSSION

If one of your part's embedded frames propagates unhandled events to your part, your part has the opportunity to handle those events. In that case, your

part's event handler needs to determine if a particular frame is one of its display frames or an embedded frame.

SEE ALSO

The `ODFrame::SetPropagateEvents` method (page 329).

DrawActiveBorder

The `DrawActiveBorder` method updates the active frame border of this frame.

```
void DrawActiveBorder ();
```

DISCUSSION

OpenDoc calls this method. This method in turn calls the `DrawActiveBorder` method associated with each of this frame's facets.

SEE ALSO

The `ODFacet::DrawActiveBorder` method (page 238).
The `ODFrame::InvalidateActiveBorder` method (page 318).

EditInLink

The `EditInLink` method returns a Boolean value that indicates whether the part maintaining a link destination (that includes this frame) can be found.

```
ODBoolean EditInLink ();
```

return value `kODTrue` if the part maintaining a link destination (that includes this frame) can be found, otherwise `kODFalse`.

DISCUSSION

Your part calls this method when the user attempts to edit in a display frame that has a link status of `kODInLinkDestination`. This method in turn calls the `EditInLinkAttempted` method of the part that maintains that link destination. That part then displays an alert allowing the user to find the source of the link or to break the link. If the user chooses to break the link, that part changes the link status of the embedded frame accordingly.

If the part maintaining the link destination was found, this method returns `kODTrue` and, if the link status of this frame was changed, you can then allow editing. In the unlikely event that the part maintaining the link destination cannot be found, this method returns `kODFalse`, and your part should display an alert informing the user that the destination of the link cannot be edited.

SEE ALSO

The `ODPart::EditInLinkAttempted` method (page 491).

FacetAdded

The `FacetAdded` method adds the facet to this frame's list of facets and notifies its part of the new facet.

```
void FacetAdded (in ODFacet facet);
```

`facet` A reference to a facet to be added.

DISCUSSION

OpenDoc calls this method when a facet is added to this frame. This method in turn calls the `FacetAdded` method associated with this frame's part to notify the part that a facet has been added to one of its display frames.

SEE ALSO

The `ODFrame::FacetRemoved` method (page 313).
The `ODPart::FacetAdded` method (page 496).

FacetRemoved

The `FacetRemoved` method removes the facet from this frame's list of facets and notifies its part of the deletion.

```
void FacetRemoved (in ODFacet facet);
```

`facet` A reference to a facet to be removed.

DISCUSSION

OpenDoc calls this method when a facet is removed. This method in turn calls the `FacetRemoved` method associated with this frame's part to notify the part that a facet has been removed from one of its display frames.

SEE ALSO

The `ODFacet::RemoveFacet` method (page 249).
The `ODFrame::FacetAdded` method (page 312).
The `ODPart::FacetRemoved` method (page 497).

GetContentExtent

The `GetContentExtent` method gets the content extent of this frame.

```
void GetContentExtent (out ODPoint contentExtent);
```

`contentExtent`
The content extent of this frame.

DISCUSSION

Content extent is, in essence, the page height of the part displayed in this frame.

SEE ALSO

The `ODPoint` type (page 855).
The `ODFrame::ChangeContentExtent` method (page 299).

GetFrameGroup

The `GetFrameGroup` method returns the group ID of this frame.

```
ODULong GetFrameGroup ();
```

return value The group ID of this frame, expressed as an unsigned 32-bit value.

SEE ALSO

The `ODFrame::SetFrameGroup` method (page 326).

GetLinkStatus

The `GetLinkStatus` method returns the link status of this frame.

```
ODLinkStatus GetLinkStatus ();
```

return value The link status of this frame. The return value is one of the following: `kODInLinkDestination`, `kODInLinkSource`, or `kODNotInLink`.

DISCUSSION

The value `kODInLinkDestination` for the `status` parameter indicates that the frame is embedded in the destination of a link; the content of this frame is thus supplied by a link. The value `kODInLinkSource` indicates that the frame is embedded in content that is the source of one or more links, but not in

content that is the destination of a link. The value `kODNotInLink` indicates that the frame is not embedded in any linked content, source or destination.

Your part calls its display frame's `GetLinkStatus` method to determine if it should allow links to be created to or from the content displayed by this frame. For example, if the link status value is `kODInLinkDestination`, a link should not be created within this frame.

SEE ALSO

The `ODFrame::ChangeLinkStatus` method (page 302).

GetPartInfo

The `GetPartInfo` method returns the part info data for this frame.

```
ODInfoType GetPartInfo ();
```

return value The part info data for this frame.

DISCUSSION

Your part calls its display frame's `GetPartInfo` method to retrieve any part-specific information it has stored there.

You should cast the return value to a pointer to your part's own representation of the data.

SEE ALSO

The `ODInfoType` type (page 853).
The `ODFacet::GetPartInfo` method (page 243).
The `ODFrame::SetPartInfo` method (page 328).

GetPresentation

The `GetPresentation` method returns this frame's presentation.

```
ODTypeToken GetPresentation ();
```

return value A tokenized string representing this frame's presentation, expressed as a 32-bit value.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODFrame::ChangePresentation` method (page 304).
The `ODFrame::SetPresentation` method (page 328).

GetSequenceNumber

The `GetSequenceNumber` method returns the sequence number of this frame in its frame group.

```
ODULong GetSequenceNumber ();
```

return value The sequence number of this frame, expressed as an unsigned 32-bit value.

SEE ALSO

The `ODFrame::ChangeSequenceNumber` method (page 305).

GetViewType

The `GetViewType` method returns this frame's view type.

```
ODTypeToken GetViewType ();
```

return value A tokenized string representing this frame's view type.

DISCUSSION

The return value must be the tokenized form of one of the view-type constants (`kODViewAsFrame`, `kODViewAsSmallIcon`, `kODViewAsLargeIcon`, or `kODViewAsThumbnail`). You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODFrame::ChangeViewType` method (page 306).
The `ODFrame::SetViewType` method (page 330).
The `ODSession::Tokenize` method (page 598).

Invalidate

The `Invalidate` method marks the specified area in this frame as in need of updating.

```
void Invalidate (in ODShape invalidShape,
                 in ODCanvas biasCanvas);
```

`invalidShape`
A reference to the shape object defining the area in this frame that needs updating, expressed in frame coordinates.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

Your part calls this method to explicitly invalidate a portion of its display frame. This method in turn calls the `Invalidate` method associated with each

of this frame's facets. The resulting invalid shape is transformed and clipped to the coordinate space of each facet's canvas.

SEE ALSO

The `ODCanvas::Invalidate` method (page 74).
The `ODFacet::Invalidate` method (page 245).
The `ODFrame::Validate` method (page 332).

InvalidateActiveBorder

The `InvalidateActiveBorder` method marks the active frame border of this frame as in need of updating.

```
void InvalidateActiveBorder ();
```

DISCUSSION

OpenDoc calls this method. This method in turn calls the `InvalidateActiveBorder` method of each of this frame's facets.

SEE ALSO

The `ODFacet::InvalidateActiveBorder` method (page 246).
The `ODFrame::DrawActiveBorder` method (page 311).

IsDragging

The `IsDragging` method returns a Boolean value that indicates whether this frame is currently being dragged.

```
ODBoolean IsDragging ();
```

return value `kODTrue` if this frame is currently being dragged, otherwise `kODFalse`.

DISCUSSION

OpenDoc calls this method to determine whether parts can be dropped onto this frame.

SEE ALSO

The `ODFrame::IsDroppable` method (page 319).
The `ODFrame::SetDragging` method (page 325).
The `ODFrame::SetDroppable` method (page 326).

IsDroppable

The `IsDroppable` method returns a Boolean value that indicates whether this frame's part accepts drag-and-drop events in this frame.

```
ODBoolean IsDroppable ();
```

return value `kODTrue` if this frame's part accepts drag-and-drop events in this frame, otherwise `kODFalse`.

DISCUSSION

OpenDoc calls this method before calling your part's `DragEnter` method to ensure that your part's display frame is able to receive a drop.

SEE ALSO

The `ODFrame::IsDragging` method (page 318).
The `ODFrame::SetDragging` method (page 325).
The `ODFrame::SetDroppable` method (page 326).

IsFrozen

The `IsFrozen` method returns a Boolean value that indicates whether this frame is bundled.

```
ODBoolean IsFrozen ();
```

return value `kODTrue` if this frame is bundled, otherwise `kODFalse`.

SEE ALSO

The `ODFrame::SetFrozen` method (page 327).

IsInLimbo

The `IsInLimbo` method returns a Boolean value that indicates whether this frame is removed from a part's content model.

```
ODBoolean IsInLimbo ();
```

return value `kODTrue` if this frame is removed from a part's content model, or `kODFalse` if this frame is in a part's content model (such as an embedded frame).

DISCUSSION

Your part calls this method to check whether this frame is in limbo.

The in-limbo flag is not a persistent property of frames; frames that are in limbo are eventually removed. When a frame object is created by a draft, the frame is not in limbo. If this frame is removed from a part's content model, such as by the Cut operation, this frame is in limbo, and eventually this frame is removed from the draft by the part that last contained the frame in its content model. If this frame is inserted into a part's content model, such as by a Paste or Undo of the cut operation, this frame is again not in limbo.

SEE ALSO

The `ODFrame::SetInLimbo` method (page 327).

IsOverlaid

The `IsOverlaid` method returns a Boolean value that indicates whether this frame is an overlaid frame.

```
ODBoolean IsOverlaid ();
```

return value `kODTrue` if this frame is an overlaid frame, otherwise `kODFalse`.

DISCUSSION

This method's return value is defined for the lifetime of this frame; once set, it cannot be changed.

IsRoot

The `IsRoot` method returns a Boolean value that indicates whether this frame is the root frame.

```
ODBoolean IsRoot ();
```

return value `kODTrue` if this frame is the root frame, otherwise `kODFalse`.

DISCUSSION

This method's return value is defined for the lifetime of this frame; once set, it cannot be changed.

SEE ALSO

The `ODFrame::IsSubframe` method (page 322).

IsSubframe

The `IsSubframe` method returns a Boolean value that indicates whether this frame is a subframe of its containing frame.

```
ODBoolean IsSubframe ();
```

return value `kODTrue` if this frame is a subframe of its containing frame, otherwise `kODFalse`.

DISCUSSION

This method's return value is defined for the lifetime of this frame; once set, it cannot be changed.

SEE ALSO

The `ODFrame::IsRoot` method (page 321).
The `ODFrame::SetSubframe` method (page 330).

Remove

The `Remove` method prepares this frame to be removed both from memory and persistent storage.

```
void Remove ();
```

DISCUSSION

Your part calls its embedded frame's `Remove` method when it permanently removes the frame from its content. This method in turn calls the

`DisplayFrameRemoved` method of this frame's part to notify the part that this frame is being removed. During execution of the `DisplayFrameRemoved` method, this frame's part calls its embedded frames's `Remove` method recursively.

During execution of this method, this frame releases its references to the associated part and its containing frame, and this frame should not be used again. After this method executes successfully, you do not need to call your part's `Release` method.

Depending on whether you want to affect persistent storage, your part calls either this method or its frame's `Close` method.

EXCEPTIONS

`kODErrFrameHasFacets` The specified frame has attached facets.

SEE ALSO

The `ODFrame::Close` method (page 307).
The `ODPart::DisplayFrameRemoved` method (page 481).

RequestFrameShape

The `RequestFrameShape` method requests a new frame shape for this frame.

```
ODShape RequestFrameShape (in ODShape shape,
                           in ODCanvas biasCanvas);
```

`shape`	A reference to the requested shape, expressed in frame coordinates.
`biasCanvas`	`kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.
return value	A reference to the new frame shape, expressed in frame coordinates.

DISCUSSION

Your part calls its display frame's `RequestFrameShape` method when it wants to resize the display frame. This method in turn calls the `RequestFrameShape` method of this frame's containing part. The containing part returns a reference to the shape object it allows the display frame to have. This frame stores the shape as its new frame shape and returns the shape to your part so that your part knows what the new shape is.

Your part must not modify the frame shape; only this frame's containing part can modify the frame shape.

This method increments the reference count of the returned shape object. When you have finished using that shape object, you should call its `Release` method.

EXCEPTIONS

`kODErrIllegalNullShapeInput`
The `shape` parameter is null.

SEE ALSO

The `ODFrame::ChangeFrameShape` method (page 300).
The `ODFrame::AcquireFrameShape` method (page 296).
The `ODPart::RequestFrameShape` method (page 523).

SetContainingFrame

The `SetContainingFrame` method assigns the specified frame as the containing frame of this frame.

```
void SetContainingFrame (in ODFrame frame);
```

`frame` A reference to the frame to assign as this frame's containing frame.

DISCUSSION

You should remove any existing facets of this frame before calling this method. It is necessary to call this method only when a frame is moved, because a frame is always created with the correct containing frame.

EXCEPTIONS

`kODErrFrameHasFacets` The specified frame has attached facets.

SEE ALSO

The `ODFrame::AcquireContainingFrame` method (page 295).

SetDragging

The `SetDragging` method specifies whether this frame is currently being dragged.

```
void SetDragging (in ODBoolean isDragging);
```

`isDragging`
`kODTrue` if this frame is currently being dragged, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to indicate whether parts can be dropped onto this frame.

SEE ALSO

The `ODFrame::IsDragging` method (page 318).
The `ODFrame::IsDroppable` method (page 319).
The `ODFrame::SetDroppable` method (page 326).

SetDroppable

The `SetDroppable` method specifies whether this frame's part accepts drag-and-drop events in this frame.

```
void SetDroppable (in ODBoolean isDroppable);
```

`isDroppable`
`kODTrue` if this frame's part accepts drag-and-drop events in this frame, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to define its display frame as either capable or incapable of accepting a drop.

SEE ALSO

The `ODFrame::IsDragging` method (page 318).
The `ODFrame::IsDroppable` method (page 319).
The `ODFrame::SetDragging` method (page 325).

SetFrameGroup

The `SetFrameGroup` method assigns a group ID to this frame.

```
void SetFrameGroup (in ODULong groupID);
```

`groupID` The group ID to assign to this frame, expressed as an unsigned 32-bit value.

DISCUSSION

Your part calls its embedded frame's `SetFrameGroup` method to change its group ID.

SEE ALSO

The `ODFrame::GetFrameGroup` method (page 314).

SetFrozen

The `SetFrozen` method specifies whether this frame is currently bundled.

```
void SetFrozen (in ODBoolean isFrozen);
```

`isFrozen` `kODTrue` if this frame is currently bundled, otherwise `kODFalse`.

DISCUSSION

Your part calls this method for any embedded frame or display frame.

SEE ALSO

The `ODFrame::IsFrozen` method (page 320).

SetInLimbo

The `SetInLimbo` method specifies whether this frame is to be removed from a part's content model.

```
void SetInLimbo (in ODBoolean isInLimbo);
```

`isInLimbo` `kODTrue` if this frame is to be removed from a part's content model, or `kODFalse` if this frame is currently in a part's content model (such as an embedded frame).

DISCUSSION

Your part calls this method to specify whether this frame is in limbo.

The in-limbo flag is not a persistent property of frames; frames that are in limbo are eventually removed. If this frame is in limbo, then this frame is removed from the draft by the part that last contained the frame in its content model. This method does not need to recursively set the flags of embedded frames.

SEE ALSO

The `ODFrame::IsInLimbo` method (page 320).

SetPartInfo

The `SetPartInfo` method assigns part info data to this frame.

```
void SetPartInfo (in ODInfoType partInfo);
```

`partInfo` The data for this frame's part info.

DISCUSSION

Your part calls its display frame's `SetPartInfo` method.

SEE ALSO

The `ODInfoType` type (page 853).
The `ODFacet::SetPartInfo` method (page 250).
The `ODFrame::GetPartInfo` method (page 315).

SetPresentation

The `SetPresentation` method assigns the specified presentation to this frame.

```
void SetPresentation (in ODTypeToken presentation);
```

`presentation`
A tokenized string representing the presentation to assign to this frame, expressed as a 32-bit value.

DISCUSSION

Your part calls its display frame's `SetPresentation` method.

EXCEPTIONS

`kODErrIllegalNullTokenInput`
The `presentation` parameter is null.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODFrame::ChangePresentation` method (page 304).
The `ODFrame::GetPresentation` method (page 316).

SetPropagateEvents

The `SetPropagateEvents` method specifies whether this frame should propagate unhandled events to its containing frame.

```
void SetPropagateEvents (in ODBoolean doesPropagateEvents);
```

`doesPropagateEvents`
`kODTrue` if this frame should propagate unhandled events to its containing frame, otherwise `kODFalse`.

DISCUSSION

Your part calls its embedded frame's `SetPropagateEvents` method to indicate whether it wants to receive events not handled by the embedded frame.

SEE ALSO

The `ODFrame::DoesPropagateEvents` method (page 310).

SetSubframe

The `SetSubframe` method specifies whether this frame is currently a subframe of its containing frame.

```
void SetSubframe (in ODBoolean isSubframe);
```

`isSubframe` `kODTrue` if this frame is currently a subframe of its containing frame, otherwise `kODFalse`.

SEE ALSO

The `ODFrame::IsSubframe` method (page 322).

SetViewType

The `SetViewType` method assigns the specified view type to this frame.

```
void SetViewType (in ODTypeToken viewType);
```

`viewType` A tokenized string representing the view type to assign to this frame.

DISCUSSION

The `viewType` parameter must be the tokenized form of one of the view-type constants (`kODViewAsFrame`, `kODViewAsSmallIcon`, `kODViewAsLargeIcon`, or `kODViewAsThumbnail`). You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

Your part calls its display frame's `SetViewType` method.

EXCEPTIONS

`kODErrIllegalNullTokenInput` The `viewType` parameter is null.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODFrame::ChangeViewType` method (page 306).
The `ODFrame::GetViewType` method (page 316).
The `ODSession::Tokenize` method (page 598).

SetWindow

The `SetWindow` method assigns a window to this frame.

```
void SetWindow (in ODWindow window);
```

`window` A reference to a window to assign to this frame.

DISCUSSION

OpenDoc calls this method when it creates this frame as its root frame. Only the root frame of a window has a reference to the window; all its embedded frames inherit this value.

EXCEPTIONS

`kODErrNotRootFrame` The specified frame is not a root frame of this part.

SEE ALSO

The `ODFrame::AcquireWindow` method (page 299).

Validate

The `Validate` method marks the specified area in this frame as no longer in need of updating.

```
void Validate (in ODShape validShape,
               in ODCanvas biasCanvas);
```

`validShape` A reference to the shape object defining the area in this frame that no longer needs updating, expressed in frame coordinates.

`biasCanvas` `kODNULL` if the geometry is in platform-normal coordinates, otherwise a reference to the canvas object to whose coordinate space the geometry is biased.

DISCUSSION

Your part calls this method to explicitly mark portions of its display frame that do not need updating. This method in turn calls the `Validate` method associated with each of this frame's facets. The `Validate` method transforms and clips the shape from the coordinate space of the facet to the coordinate space of its canvas and subtracts the shape from any existing invalid area of the canvas.

SEE ALSO

The `ODCanvas::Validate` method (page 80).
The `ODFacet::Validate` method (page 252).
The `ODFrame::Invalidate` method (page 317).

ODFrameFacetIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODFrameFacetIterator` class provides access to all facets of a frame.

Description

You use a frame-facet iterator to apply an operation to all facets of one of your part's display frames, such as drawing and changing their shapes. For example, a part might use a frame-facet iterator to delete all facets of an embedded frame that it deletes, or to update all views of one of its display frames if it were displaying asynchronously.

Your part creates a frame-facet iterator object by calling its frame's `CreateFacetIterator` method (page 308), which returns a reference to a frame-facet iterator object.

While you are using a frame-facet iterator, you should not modify the list of facets for the frame. You must postpone adding facets to or removing facets from the list of facets for the frame until after you have deleted the iterator.

For more information related to facet objects, see the `ODFacet` class description (page 215). For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODFrameFacetIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and returns a reference to the first facet in the iteration sequence.
`Next`	Returns a reference to the next facet in the iteration sequence.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and returns a reference to the first facet in the iteration sequence.

```
ODFacet First ();
```

return value A reference to the first facet in the iteration sequence, or `kODNULL` if the frame has no facets.

DISCUSSION

Your part must call this method before calling this frame-facet iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

EXCEPTIONS

`kODErrIteratorOutOfSync`
The list of facets for the frame was modified while the iteration was in progress.

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more facets remain in the iteration sequence. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found a facet. This method returns `kODFalse` when you have examined all the facets.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The list of facets for the frame was modified while the iteration was in progress.

Next

The `Next` method returns a reference to the next facet in the iteration sequence.

```
ODFacet Next ();
```

return value A reference to the next facet in the iteration sequence, or `kODNULL` if you have reached the last facet.

DISCUSSION

If your part calls this method before calling this frame-facet iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of facets for the frame was modified while the iteration was in progress.

ODInfo

Superclasses `ODObject`

Subclasses none

An object of the `ODInfo` class provides the Part Info dialog box, which your part uses to display standard user properties about itself or its embedded parts.

Description

When a document is opened, the session object creates a single info object. All parts of that document share the info object; you can obtain a reference to it by calling the session object's `GetInfo` method (page 584).

The info object allows you to see the Part Info dialog box for a part. That dialog box displays properties of the part that are of interest to the user, for example, its part kind and category, size, and creation date. If you have implemented a settings extension for your part, the dialog box includes a Settings button that the user can click to see additional properties specific to your part. When the user clicks the Settings button, OpenDoc calls your settings extension's `ShowSettings` method (page 605).

For additional information about the Part Info dialog box and the settings extension, see the chapters on windows and menus and extending OpenDoc in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents a summary description of the `ODInfo` method, followed by a detailed description.

`ShowPartFrameInfo` Displays the Part Info dialog box in response to a user selection from the Edit menu.

ShowPartFrameInfo

The `ShowPartFrameInfo` method displays the Part Info dialog box in response to a user selection from the Edit menu.

```
ODBoolean ShowPartFrameInfo (in ODFacet facet,
                             in ODBoolean allowEditing);
```

`facet` A reference to the facet that indicates the window to display the Part Info dialog box in. The facet's frame and part indicate which part's user properties should be displayed and edited.

`allowEditing`
`kODTrue` if the part allows editing, otherwise `kODFalse`.

return value `kODTrue` if the user clicked OK to leave the Part Info dialog box, otherwise `kODFalse`.

DISCUSSION

You call this method in your part's `HandleEvent` method when the user selects the Part Info item from the Edit menu and the current selection is an embedded frame border.

SEE ALSO

The `ODPart::HandleEvent` method (page 506).

ODLink

Superclasses `ODPersistentObject` → `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODLink` class represents the destinations of an OpenDoc link. Instances of this class are created and maintained by draft objects whenever the user creates a link between parts.

Description

Linking is a mechanism for associating data in one part (the **source**) with data in another location (the **destination**) in such a way that the destination data can be updated either manually or automatically whenever the source data changes. A typical example of linking allows the user to paste a spreadsheet graph into a financial report in such a way that subsequent changes to the spreadsheet are automatically reflected in the report.

Links are created during paste or drop operations. The source and destination can be in the same part, in different parts in the same document, or in different documents. A link is a persistent, one-way conduit; updating occurs from the source to the destination only. Link sources are represented by `ODLinkSource` objects; link destinations are represented by `ODLink` objects.

When the user requests a link, the following events occur:

- The destination part asks for a link object by calling its draft's `AcquireLink` method (page 152).
- If the source part is in the same document, the draft calls the source part's `CreateLink` method (page 476). The source part may refuse this request, in which case the link cannot be created; normally, however, the source part would refuse only in the event of an error.

  If the source part is in a different document, the link manager creates a cross-document link.

- If the link-source object was created successfully, the draft's `AcquireLink` method returns a link object to the destination part.

Every link-source object has an associated update ID that uniquely identifies the current generation or version of its content. The source part sets the update ID whenever it creates or updates the content of a link-source object. The destination part stores the ID that was current when it last updated its content from the link.

The destination part reads link data from a link object's content storage unit. Because source and destination parts may attempt to access a link simultaneously, parts must acquire a lock before they can access the content storage unit. You can lock a link object by calling its `Lock` method (page 345); you can then obtain a reference to the link object's content storage unit by calling its `GetContentStorageUnit` method (page 343). You must not cache that storage unit; instead you must call the `GetContentStorageUnit` method whenever you need to access the storage unit.

A destination part can register itself for automatic notification of updates by calling the `RegisterDependent` method (page 346) of the link object. Whenever the link-source object sends updates to the link object, the link object calls the `LinkUpdated` method (page 512) of each of its registered destination parts to notify those parts to read the new data from the link object's content storage unit.

A destination part is responsible for updating its destination content from the link object. If your part is the destination part of a link, its `LinkUpdated` method should read data from the link object's content storage unit and use that data to update its destination content. In addition, after your part displays the Show Link Destination Info dialog box, it should update its destination content if the user exits the dialog box by clicking the Update Now button.

Each link destination tracks a single link source, but there can be several link destinations for any given link source. Within a given draft, a single link object is associated with each link-source object, even if the source is linked to multiple destinations in the draft; all destinations in the draft share this link object. For this reason, a part that maintains two or more destinations of a given link source must be careful to register only once with the corresponding link object.

A destination part's registration with a link is not permanent; the part should reregister when it is re-created from its stored data if the destination content is visible or if it could affect the layout of the part. A destination part may

explicitly unregister itself from a link by calling the `UnregisterDependent` method (page 351) of the link object. A destination part with a single destination for a particular link source should unregister when the user breaks the link at the destination, when the linked content is deleted, or when the user changes the destination from automatic to manual updating. A destination part with multiple destinations of the same link source should unregister in the same situations, provided that none of the other destinations gets updates automatically.

If your part contains a link destination, you are responsible for drawing an appropriate border around the linked content when requested to do so. If the user selects any content within the link, or checks the Show Links setting of the Document Info dialog box, you must show the border of the link destination whenever you draw. You can check whether to show links by calling the window's `ShouldShowLinks` method (page 809) before drawing your part's content.

For further information on the implementation of OpenDoc links, see the descriptions of the companion classes `ODLinkSource` (page 361) and `ODLinkSpec` (page 379) and the chapter on data transfer in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODLink` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [M] are specific to the Mac OS platform.

Link Access

`Lock` Locks this link object, ensuring exclusive read-only access to its content storage unit.

`Unlock` Unlocks this link object, relinquishing access to its content storage unit.

`GetContentStorageUnit` Returns a reference to the storage unit containing the content of this link object.

Updating

`GetUpdateID`	Returns the update ID that uniquely identifies the current generation or version of the link source content.
`GetChangeTime`	Returns the time of the last update to the link source content.
`RegisterDependent`	Puts the specified destination part on the list of parts to be notified whenever the link is updated.
`UnregisterDependent`	Removes the previously registered destination part from the list of parts to be notified whenever the link is updated.

Link Information

`ShowSourceContent`	Requests the source part of the link to make the source content visible.
`ShowLinkDestinationInfo` [M]	Displays the Link Destination Info dialog box for this link object.

GetChangeTime

The `GetChangeTime` method returns the time of the last update to the link source content.

```
ODTime GetChangeTime ();
```

return value The time when the link source content was last updated.

SEE ALSO

The `ODTime` type (page 847).

GetContentStorageUnit

The `GetContentStorageUnit` method returns a reference to the storage unit containing the content of this link object.

```
ODStorageUnit GetContentStorageUnit (in ODLinkKey key);
```

`key`	A valid link key obtained by a prior call to the `Lock` method.
return value	A reference to this link object's content storage unit.

DISCUSSION

The `key` parameter ensures thread safety. Before calling this method, you must call this link object's `Lock` method to obtain this key. The returned storage unit remains valid until the key is relinquished by the `Unlock` method.

You may read and copy, but should not change, the data in the returned content storage unit. The link object handles the creation and destruction of its content storage unit, so you must neither dispose of nor release the returned storage unit.

You must not cache the returned storage unit; instead you must call this method whenever you need to access the content storage unit.

EXCEPTIONS

`kODErrBrokenLink`	Internal error; the link-source object disconnected from its destinations.
`kODErrCannotEstablishLink`	A persistent link could not be established.
`kODErrInvalidLinkKey`	The `key` parameter is not a valid key for this link.
`kODErrNoLinkContent`	The content storage unit has no contents property.

SEE ALSO

The `ODLinkKey` type (page 894).
The `ODStorageUnit` class (page 641).

GetUpdateID

The `GetUpdateID` method returns the update ID that uniquely identifies the current generation or version of the link source content.

```
ODUpdateID GetUpdateID ();
```

return value The current update ID of the link source content.

DISCUSSION

If your part contains a destination of this link, you can call this method to determine the current version of the content of the link-source object. You can compare the returned update ID with a previously saved ID to establish whether the source content has been modified since you last read it. There is no implicit ordering of update ID values: they offer tests for equality only, with no indication of the time or nature of any link changes.

You can arrange for your part to receive automatic notification of content updates by calling the `RegisterDependent` method.

EXCEPTIONS

`kODErrBrokenLink`	Internal error; the link-source object disconnected from its destinations.

SEE ALSO

The `ODUpdateID` type (page 887).
The `ODLink::RegisterDependent` method (page 346).
The `ODLinkSource::ContentUpdated` method (page 368).

Lock

The `Lock` method locks this link object, ensuring exclusive read-only access to its content storage unit.

```
ODBoolean Lock (in ODULong wait,
                out ODLinkKey key);
```

`wait`	The interval to wait for access to be granted.
`key`	If access is granted, a valid link key; otherwise an undefined, invalid key.
return value	`kODTrue` if access is granted, otherwise `kODFalse`.

DISCUSSION

To ensure thread-safe access, you must call this method to acquire a valid link key before you can read the link data. This method grants read-only access; a destination part cannot modify a link's content.

The `wait` parameter specifies the time you are willing to wait for access to be granted. A value of 0 means no wait and is the only value accepted on the Mac OS platform. Other platforms may accept other values with platform-dependent meanings.

A link may be locked by only one object at a time; nested calls to the `Lock` method deny access.

While your part has the link locked, you must pass the key returned in the `key` output parameter to all methods that access the link. When you are finished using the link, you must pass this key to the `Unlock` method to unlock the link.

EXCEPTIONS

`kODErrBrokenLink`	Internal error; the link-source object disconnected from its destinations.

SEE ALSO

The `ODLinkKey` type (page 894).
The `ODLink::GetContentStorageUnit` method (page 343).
The `ODLink::Unlock` method (page 351).

RegisterDependent

The `RegisterDependent` method puts the specified destination part on the list of parts to be notified whenever the link is updated.

```
void RegisterDependent (in ODPart clientPart,
                        in ODUpdateID id);
```

`clientPart` A reference to the destination part wishing to be notified of changes to the content of the link source.

`id` The update ID of the content the specified part last read from the link source.

DISCUSSION

If your part is a destination of this link, you can call this method to register your part as a dependent of this link object. Dependent parts receive automatic notification of any changes to the link's content. You can remove your part from the list of dependents by calling the `UnregisterDependent` method.

You should call this method in your part's `HandleEvent` method after calling the `ShowLinkDestinationInfo` method if the user has requested automatic notification of updates and your part is not already a registered dependent of this link object.

If the link's current update ID differs from the `id` parameter, OpenDoc immediately calls the specified client part's `LinkUpdated` method. You should pass the constant `kODUnknownUpdate` as the value of the `id` parameter when the link is first created; doing so ensures that your part's `LinkUpdated` method is called.

IMPORTANT

You should not call this method if your part is already a registered dependent of this link object, for example, because your part contains another destination of link source corresponding to this link object. ▲

EXCEPTIONS

`kODErrBrokenLink`	Internal error; the link-source object disconnected from its destinations.
`kODErrCannotRegisterDependent`	The link object is unable to register the specified part as a dependent at this time.

SEE ALSO

The `ODUpdateID` type (page 887).
The `ODLink::GetUpdateID` method (page 344).
The `ODLink::ShowLinkDestinationInfo` method (page 347).
The `ODLink::UnregisterDependent` method (page 351).
The `ODPart::LinkUpdated` method (page 512).

ShowLinkDestinationInfo

Mac OS

The `ShowLinkDestinationInfo` method displays the Link Destination Info dialog box for this link object.

```
ODBoolean ShowLinkDestinationInfo (
                        in ODFacet facet,
                        in ODLinkInfo info,
                        out ODLinkInfoResult infoResult);
```

`facet`	A reference to the facet displaying the selected link destination.
`info`	A structure containing link destination information.

`infoResult`	A structure reflecting the user's selections in the Link Destination Info dialog box.
return value	`kODTrue` if the user did not cancel the Link Destination Info dialog box, otherwise `kODFalse`.

DISCUSSION

You call this method in your part's `HandleEvent` method when the user selects the Link Info item from the Edit menu and the current selection is the border of a link destination.

The Link Destination Info dialog box displays the part kind of the linked data, together with the dates and times of its creation and last update. The dialog box lets the user set the link update mode to automatic or manual. The user can also break the link, request an immediate update from the link data, or display the link source for editing.

If the user exits the Link Destination Info dialog box by clicking a button other than Cancel, this method returns true and you should examine the `action` field of the `infoResult` output parameter to determine what action to take in response to the user's selections.

- If the `action` field is `kODLinkInfoFindSource`, you should call this link objects's `ShowSourceContent` method.
- If the `action` field is `kODLinkInfoBreakLink`, you should no longer associate your part's destination content with this link. If the selected link destination gets updates automatically and your part has no other destination of this link object that also gets updates automatically, you should call the `UnregisterDependent` to unregister your part.
- This operation should be undoable.
- If the `action` field is `kODLinkInfoUpdateNow`, you should update its content from this link.
- If the `action` field is `kODLinkInfoOk`, you should examine the `autoUpdate` field of the `infoResult` output parameter. If the `autoUpdate` field has changed, you should change your part's automatic notification status as appropriate.
  - If the user selected manual notification and your part has no other destination of this link object that gets updates automatically, you should call the `UnregisterDependent` method to unregister your part.

- If the user selected automatic notification and your part isn't already registered to receive automatic notification (because some other destination of this link object gets automatic updates), you should call the `RegisterDependent` method to register your part.

If the user cancels the dialog box, this method returns false and you do not need to take any further action.

EXCEPTIONS

`kODErrBrokenLink`	Internal error; the link-source object disconnected from its destinations.
`kODErrNullFacetInput`	The `facet` parameter is null.
`kODErrNullLinkInfoInput`	The `info` parameter is null.
`kODErrNullLinkInfoResultInput`	The `infoResult` parameter is null.

SEE ALSO

The `ODLinkInfo` type (page 893).
The `ODLinkInfoAction` type (page 893).
The `ODLinkInfoResult` type (page 894).
The `ODLink::RegisterDependent` method (page 346).
The `ODLink::ShowSourceContent` method (page 349).
The `ODLink::UnregisterDependent` method (page 351).
The `ODLinkSource::ShowLinkSourceInfo` method (page 375).

ShowSourceContent

The `ShowSourceContent` method requests the source part of the link to make the source content visible.

```
void ShowSourceContent ();
```

DISCUSSION

If your part is the destination of this link, you should call this method in your part's `HandleEvent` method after calling the `ShowLinkDestinationInfo` method if the latter method returns true and its output parameter indicates that the user requested to see the source content. You also should call this method in your part's `EditInLinkAttempted` method if the user elects to find the source of the link.

This method causes the source part's `RevealLink` method to be called, making the source content visible.

If this method throws an exception, you should display an alert message informing the user that the link source content could not be shown.

EXCEPTIONS

`kODErrBrokenLink` Internal error; the link-source object disconnected from its destinations.

`kODErrBrokenLinkSource` The link has been broken at the source.

`kODErrCannotFindLinkSource` Cannot find the source of this cross-document link.

`kODErrCannotFindLinkSourceEdition` The edition file for this cross-document link is missing (the link may have been broken at the source).

This method may also throw exceptions that were raised by the `ODPart::RevealLink` method.

SEE ALSO

The `ODLink::ShowLinkDestinationInfo` method (page 347).
The `ODPart::EditInLinkAttempted` method (page 491).
The `ODPart::RevealLink` method (page 526).

Unlock

The `Unlock` method unlocks this link object, relinquishing access to its content storage unit.

```
void Unlock (in ODLinkKey key);
```

`key` A valid link key obtained by a prior call to the `Lock` method.

DISCUSSION

You should call this method as soon as possible after accessing the content storage unit of this link object.

The `key` parameter should be a valid key obtained by an earlier call to the `Lock` method. After this method executes successfully, the specified key is no longer valid and access to the link's content is relinquished.

EXCEPTIONS

`kODErrBrokenLink`	Internal error; the link-source object disconnected from its destinations.
`kODErrInvalidLinkKey`	The `key` parameter is not a valid key for this link.

SEE ALSO

The `ODLinkKey` type (page 894).
The `ODLink::Lock` method (page 345).

UnregisterDependent

The `UnregisterDependent` method removes the previously registered destination part from the list of parts to be notified whenever the link is updated.

```
void UnregisterDependent (in ODPart clientPart);
```

`clientPart` A reference to the destination part to be unregistered.

DISCUSSION

If your part is the destination of this link, you can call this method to unregister your part as a dependent. Dependent parts receive automatic notification of any changes to the link's content. You can reregister your part by calling the `RegisterDependent` method.

You should call this method in your part's `HandleEvent` method after calling the `ShowLinkDestinationInfo` method if the user has broken the link or has changed from automatic to manual notification. In either case, you should call this method only if the selected link destination is your part's only registered destination of the link-source object corresponding to this link object.

The `UnregisterDependent` method returns without error if the specified destination part was not previously registered as a dependent of this link.

SEE ALSO

The `ODLink::RegisterDependent` method (page 346).
The `ODLink::ShowLinkDestinationInfo` method (page 347).

ODLinkManager

Superclasses `ODObject`

Subclasses none

An object of the `ODLinkManager` class coordinates the creation and maintenance of cross-document links.

Description

The `ODLinkManager` class uses a platform-dependent implementation of cross-document links, building on facilities provided on each platform. On the Mac OS platform, for example, the link manager uses the Mac OS Edition Manager.

When a document is opened, the session object creates a single link-manager object. All parts of that document share the link manager. Parts do not access the link manager directly; instead, source parts use the `ODLinkSource` class (page 361) and destination parts use the `ODLink` class (page 339).

The document shell or a container application can obtain a reference to the link manager by calling the session object's `GetLinkManager` method (page 585). The link manager's methods are called by link objects, draft objects, and document shells; they are not used by parts.

When a cross-document link is established, the link-manager object of the destination draft communicates with the link-manager object of the source draft to create the link. Neither the source nor destination parts know that the link is cross-document; the source draft and the destination draft each maintain a link-source object and a link object, and the parts involved communicate only with the link-source objects and link objects of their own drafts.

When a draft is opened, the link-manager object ensures that the link destinations from other documents have the opportunity to update to the latest content. It also maintains a list of cross-document link sources that should be updated when the draft is saved.

On the Mac OS platform, the link-manager object provides the coordination of Edition Manager section IDs with the OpenDoc shell or with container applications that support Publish and Subscribe. A container application can reserve section IDs already in use by calling the `ReserveSectionID` method (page 358). A container application can get a new section ID that is unique for the life of the document by calling the `NewSectionID` method (page 358).

For more information related to links, see the descriptions of the classes `ODLink` (page 339), `ODLinkSource` (page 361), and `ODLinkSpec` (page 379). For more information about linking, see the chapter on data transfer in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODLinkManager` methods, followed by detailed descriptions in alphabetical order. Methods marked [D] are called only by the document shell or container applications. Methods marked [M] are specific to the Mac OS platform.

`AnyLinkImported` [D] Returns a Boolean value that indicates whether one or more links to the source in other documents have been updated automatically since the specified draft was last saved.

`UnsavedExportedLinks` [D] Returns a Boolean value that indicates whether any cross-document links have been established since the last time the specified draft was saved.

`DraftClosing` [D] Notifies this link manager that the specified draft is about to be closed.

`DraftOpened` [D] Notifies this link manager that the specified draft has been opened.

`DraftSaved` [D] Notifies this link manager that the specified draft has been saved.

`NewSectionID` [D] [M] Returns a new section ID for the document of the specified draft.

`ReserveSectionID` [D] [M]
Reserves a section ID for the lifetime of the document of the specified draft.

AnyLinkImported

Document shell

The `AnyLinkImported` method returns a Boolean value that indicates whether one or more links to the source in other documents have been updated automatically since the specified draft was last saved.

```
ODBoolean AnyLinkImported (in ODDraft draft);
```

`draft`	A reference to the draft to be tested for imported links.
return value	`kODTrue` if one or more links have been automatically updated, otherwise `kODFalse`.

DISCUSSION

A container application could call this method when it might otherwise close the document without saving. If the method returns true, links were updated and the document should be saved without user interaction.

At present, the document shell does not call this method.

SEE ALSO

The `ODLinkManager::UnsavedExportedLinks` method (page 359).

DraftClosing

Document shell

The `DraftClosing` method is called to notify this link manager that the specified draft is about to be closed.

```
void DraftClosing (in ODDraft draft);
```

`draft` A reference to the draft to be closed.

DISCUSSION

The document shell calls this method before reverting or closing the specified draft. If the draft contains any cross-document links created since the last save, the link manager breaks those links at the source; on the Mac OS platform, it deletes the edition files for those links.

A container application can call the `UnsavedExportedLinks` method to determine whether closing the document without saving the draft would result in the loss of cross-document links.

SEE ALSO

The `ODLinkManager::UnsavedExportedLinks` method (page 359).

DraftOpened

Document shell

The `DraftOpened` method is called to notify this link manager that the specified draft has been opened.

```
void DraftOpened (in ODDraft draft);
```

`draft` A reference to the draft that was opened.

DISCUSSION

The document shell calls this method after the specified draft is opened. The link manager reads each link object in the draft into memory, allowing those subscribing to or publishing edition files on the Mac OS platform to register with the Edition Manager.

SEE ALSO

The `ODLinkManager::DraftClosing` method (page 356).

DraftSaved

Document shell

The `DraftSaved` method is called to notify this link manager that the specified draft has been saved.

```
void DraftSaved (in ODDraft draft);
```

`draft` A reference to the draft that was saved.

DISCUSSION

The document shell calls this method after saving a draft. Following each save, links that publish a cross-document link on the Mac OS platform update their edition files.

EXCEPTIONS

On the Mac OS platform, this method throws a platform-specific exception if the edition files cannot be updated. The document shell or container application should report this error to the user. In the event of such an error, the draft should be considered unsaved.

NewSectionID

Document shell
Mac OS

The `NewSectionID` method returns a new section ID for the document of the specified draft.

```
ODULong NewSectionID (in ODDraft draft);
```

`draft`	A reference to the draft in which to reserve a section ID.
return value	An ID that is unique in the specified draft and any subsequent drafts of the same document.

DISCUSSION

The returned section ID is guaranteed to be unique for the lifetime of the specified draft.

SEE ALSO

The `ODLinkManager::ReserveSectionID` method (page 358).

ReserveSectionID

Document shell
Mac OS

The `ReserveSectionID` method reserves a section ID for the lifetime of the document of the specified draft.

```
ODBoolean ReserveSectionID (in ODULong sectionID,
                            in ODDraft draft);
```

`sectionID`	The section ID to be reserved.
`draft`	A reference to the draft in which to reserve the section ID.

return value `kODTrue` if the specified section ID is not in use and not already reserved, otherwise `kODFalse`.

DISCUSSION

This method guarantees that the specified section ID will not be issued by the `NewSectionID` method for the lifetime of the document. If this method returns false, the caller should request a different ID or call the `NewSectionID` method.

SEE ALSO

The `ODLinkManager::NewSectionID` method (page 358).

UnsavedExportedLinks

Document shell

The `UnsavedExportedLinks` method returns a Boolean value that indicates whether any cross-document links have been established since the last time the specified draft was saved.

```
ODBoolean UnsavedExportedLinks (in ODDraft draft);
```

`draft` A reference to the draft of interest.

return value `kODTrue` if cross-document links have been established since the specified draft was last saved, otherwise `kODFalse`.

DISCUSSION

A container application can call this method when the user tries to close or revert a draft without first saving it. A return value of true indicates that some cross-document links will be lost unless the draft is saved, and the container application could alert the user accordingly.

At present, the document shell does not call this method.

SEE ALSO

The `ODLinkManager::AnyLinkImported` method (page 355).
The `ODLinkManager::DraftClosing` method (page 356).

ODLinkSource

Superclasses `ODPersistentObject` → `ODRefCntObject` → `ODObject`
Subclasses none

An object of the `ODLinkSource` class represents the source of an OpenDoc link. Link-source objects are created and maintained by draft objects whenever the user creates a link between parts.

Description

Linking is a mechanism for associating data in one part (the **source**) with data in another location (the **destination**) in such a way that the destination data can be updated either manually or automatically whenever the source data changes. A typical example of linking allows the user to paste a spreadsheet graph into a financial report in such a way that subsequent changes to the spreadsheet are automatically reflected in the report.

Links are created during paste or drop operations. The source and destination can be in the same part, in different parts in the same document, or in different documents. A link is a persistent, one-way conduit; updating occurs from the source to the destination only. Link sources are represented by `ODLinkSource` objects; link destinations are represented by `ODLink` objects. Source parts reference only link-source objects, and destination parts reference only link objects. These reference restrictions are important in ensuring the proper behavior when linked content is copied or moved.

When a source part that supports linking writes to the clipboard or drag-and-drop object, it uses a link specification to advertise its ability to create links. The source part creates a link specification (an object of class `ODLinkSpec`) by calling its draft's `CreateLinkSpec` method (page 164). In addition to writing the source content to the clipboard or drag-and-drop object, the source part writes link-specification data to the `kODPropLinkSpec` property. Subsequently, the user who pastes or drops the source data to a destination part can request a link by means of the Paste As dialog box.

When the user requests a link, the destination part asks for a link object by calling its draft's `AcquireLink` method (page 152). If the source part is in the same document, the draft calls the source part's `CreateLink` method (page 476). If the source part has not already created a link-source object for the source content described by the link specification, the source part calls its draft's `CreateLinkSource` method (page 163) to create a link-source object. The source part's draft is responsible for both the transitory and persistent storage of its link-source objects. If the source part is in a different document, the link manager creates a cross-document link.

When a source part is re-created from its stored data, its `InitPartFromStorage` method (page 510) can call its draft object's `AcquireLinkSource` method (page 153) to re-create the link-source object from storage.

If your part contains a link source, you are responsible for drawing an appropriate border around the linked content when requested to do so. If the user selects any content within the link, or checks the Show Links setting of the Document Info dialog box, you must show the border of the link source whenever you draw. You can check whether to show links by calling the window's `ShouldShowLinks` method (page 809) before drawing your part's content. If the user cuts content from your part that includes a link source, your part relinquishes ownership of the link-source object. If the user subsequently undoes the action, you must call the `SetSourcePart` method (page 374) of the link-source object to reclaim ownership.

Every link-source object has an associated update ID that uniquely identifies the current generation or version of its content. The source part sets the update ID whenever it creates or updates the content of the link-source object. The destination part stores the ID that was current when it last updated its content from the link.

For further information on the implementation of OpenDoc links, see the descriptions of the companion classes `ODLink` (page 339) and `ODLinkSpec` (page 379), and the chapter on data transfer in the *OpenDoc Programmer's Guide for the Mac OS*.

Creating Multiple Destinations for a Link

If your part is the source part for a link, its `CreateLink` method may be called multiple times with the same link specification to create multiple destinations

for the same link. If you write data to the content storage unit of the link-source object when you first create it, your `CreateLink` method can simply return the link-source object on subsequent calls. However, if you write promises, you need to ensure that all part kinds originally promised are present in the content storage unit. The initial destination caused your part to fulfill promises of the part kinds that it copied, but might have left unfulfilled promises for other part kinds. If the draft was saved, those unfulfilled promises were removed. To add a new destination, you need to clear the content storage unit and rewrite promises for all part kinds. First, call the link-source object's `GetUpdateID` method (page 371) to get its current update ID. Next, pass that ID as the parameter when you call the `Clear` method (page 366). Because the ID is not being changed, the link-source object keeps track of the nonpromise part kinds that were in the content storage unit when you cleared it. When you subsequently write promises for those part kinds, the link-source object automatically forces the source part to fulfill them. Thus, you can ensure that the content storage unit contains all the data needed by the initial destination part and promises for the other part kinds that might be needed by the new destination part.

Data Transfer from Source to Destination

The process of transferring content from a link source to a link destination requires three steps. First, the source part transfers content to the link-source object. Second, the link-source object transfers content to each of its associated link objects. Third, each destination part must read the content from its link object.

Update Mode

Every link-source object has an update mode that affects the first two steps of the transfer from source part to destination part. The source part should use the update mode to determine when to transfer content to the link-source object. In addition, the link-source object uses the update mode to determine when to send updated content to its associated link objects in other documents. (Regardless of the update mode, whenever the source part updates the link-source object, the link-source object immediately sends the updated content to its associated link objects in the same document.)

- The automatic update mode indicates that the source part should update the link-source object whenever the source content has changed; the link-source object sends those updates to destinations in other documents whenever its draft is saved. This mode is the default.
- The manual update mode indicates that the source part should update the link-source object only when the user clicks the Update Now button in the Link Source Info dialog box; the link-source object immediately sends those updates to destinations in other documents.

Source Part to Link-Source Object

If your part is the source part of a link, you should use the update mode of the link-source object to determine when to transfer data. Several methods are required to effect the data transfer.

Because source and destination parts may attempt to access a link simultaneously, parts must acquire a lock before they transfer link data. You can lock the link-source object by calling its `Lock` method (page 372).

Once the link-source object's is locked, you can obtain a reference to its content storage unit by calling its `GetContentStorageUnit` method (page 370). You must not cache that storage unit; instead you must call the `GetContentStorageUnit` method whenever you need to access the storage unit.

If you are updating the content, as opposed to writing initial content, you should call the link-source object's `Clear` method (page 366) before you write to the content storage unit.

You can call the appropriate methods of the content storage unit to write data to it. If you need to clone objects, call the cloning methods of your part's draft object. For more information, see the descriptions of the classes `ODStorageUnit` (page 641) and `ODDraft` (page 145).

After writing initial content or updates to the link-source object's content storage unit, you must notify the link-source object that you have updated its content by calling its `ContentUpdated` method (page 368). Finally, you should unlock the link-source object by calling its `Unlock` method (page 377).

Link-Source Object to Link Object

The link-source object automatically transfers new content to its associated link objects. This step requires no action by either the source part or the destination part. As mentioned previously, the time when this transfer occurs varies depending on the update mode of the link-source object and whether the link object is in the same document as the link-source object.

Link Object to Destination Part

Every destination part is responsible for updating its destination content from the link object. A destination part can register itself for automatic notification of updates by calling the `RegisterDependent` method (page 346) of the link object. A registered destination part should update its content from the link object when it is notified of an update; an unregistered part should update its content when the user clicks the Update Now button in the Show Link Destination Info dialog box.

Methods

This section presents summary descriptions of the `ODLinkSource` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [M] are specific to the Mac OS platform.

Link-Source Access

`Lock`	Locks this link-source object, ensuring exclusive access to its content storage unit.
`GetContentStorageUnit`	Returns a reference to the content storage unit for this link-source object.
`Unlock`	Unlocks this link-source object, relinquishing access to its content storage unit.
`SetSourcePart`	Establishes the source part that is to maintain this link-source object.

Updating

`Clear`	Clears the content of this link-source object.

`ContentUpdated`	Notifies this link-source object that its content has been updated.
`GetUpdateID`	Returns the update ID that uniquely identifies the current generation or version of this link-source object's content.
`GetChangeTime`	Returns the time when this link-source object was last updated.
`IsAutoUpdate`	Returns a Boolean value that indicates whether this link-source object's update mode is automatic.
`SetAutoUpdate`	Sets this link-source object's update mode to automatic or manual.

Link-Source Information

`ShowLinkSourceInfo` [M]
Displays the Link Source Info dialog box for this link-source object.

Clear

The `Clear` method clears the content of this link-source object.

```
void Clear (in ODUpdateID id,
            in ODLinkKey key);
```

`id`	The update ID associated with the new empty content.
`key`	A valid link key obtained by a prior call to the `Lock` method.

DISCUSSION

If your part is the source part for this link-source object, you call this method to remove the link-source object's previous content before updating the source data. This method removes all data in the contents property (`kODPropContents`) from the link-source object's content storage unit.

The `id` parameter identifies the new version of this link-source object's content; its value depends on the circumstance in which this method is called:

- If your part originated the changes to its source content, the `id` parameter should be a new update ID that you obtained by calling the `UniqueUpdateID` method of the session object.
- If the changes occurred because the source content of this link contains the destination content of another link that was updated, the `id` parameter should be the update ID that your part received when it was notified to update the destination content within the source content for this link.
- If your part originally created this link-source object with promise data and its `CreateLink` method is being called again to create a new link destination, the `id` parameter should be the current update ID for this link-source object. You can call the `GetUpdateID` method to get the current update ID.

  To add a new destination for this link-source object, you need to clear the content storage unit and rewrite promises for all part kinds. Because the ID is not being changed, the link-source object keeps track of the nonpromise part kinds that were in content storage unit when you cleared it. When you subsequently rewrite promises for those part kinds, the link-source object automatically forces the source part to fulfill them; you should *not* call the link-source object's `ContentUpdated` method after rewriting those promises.

The `key` parameter ensures thread safety. Before calling this method, you must call this link-source object's `Lock` method to obtain this key.

EXCEPTIONS

`kODErrInvalidLinkKey`	The `key` parameter is not a valid key for this link-source object.
`kODErrInvalidPermissions`	Draft permissions do not allow modifications.
`kODErrUnknownUpdateID`	The specified update ID is the reserved value `kODUnknownUpdate`.

SEE ALSO

The `ODLinkKey` type (page 894).
The `ODUpdateID` type (page 887).
The `ODLinkSource::GetContentStorageUnit` method (page 370).
The `ODLinkSource::GetUpdateID` method (page 371).
The `ODLinkSource::Lock` method (page 372).
The `ODSession::UniqueUpdateID` method (page 599).

ContentUpdated

The `ContentUpdated` method is called to notify this link-source object that its content has been updated.

```
void ContentUpdated (in ODUpdateID id,
                     in ODLinkKey key);
```

`id` The update ID associated with the new content.

`key` A valid link key obtained by a prior call to the `Lock` method.

DISCUSSION

If your part is the source part for this link-source object, you should call this method after writing the initial content to the content storage unit of this link-source object and after making changes to the content of that storage unit.

The `id` parameter identifies the new version of this link-source object's content; its value depends on the circumstance in which this method is called:

- If you called the `Clear` method before calling this method, the `id` parameter should be the same update ID that you passed to the `Clear` method.
- If your part originated the changes to its source content, the `id` parameter should be a new update ID that you to obtained by calling the `UniqueUpdateID` method of the session object.
- If the changes occurred because the source content of this link contains the destination content of another link that was updated, the `id` parameter

should be the update ID that your part received when it was notified to update the destination content within the source content for this link.

The `key` parameter ensures thread safety. Before calling this method, you must call the `Lock` method to obtain this key.

EXCEPTIONS

`kODErrInvalidLinkKey`	The `key` parameter is not a valid key for this link-source object.
`kODErrUnknownUpdateID`	The specified update ID is the reserved value `kODUnknownUpdate`.

SEE ALSO

The `ODLinkKey` type (page 894).
The `ODUpdateID` type (page 887).
The `ODFrame::ChangeLinkStatus` method (page 302).
The `ODFrame::ContentUpdated` method (page 308).
The `ODLinkSource::GetUpdateID` method (page 371).
The `ODLinkSource::Lock` method (page 372).
The `ODPart::EmbeddedFrameUpdated` method (page 494).
The `ODSession::UniqueUpdateID` method (page 599).

GetChangeTime

The `GetChangeTime` method returns the time when this link-source object was last updated.

```
ODTime GetChangeTime ();
```

return value The time when this link-source object was last updated.

SEE ALSO

The `ODTime` type (page 847).

GetContentStorageUnit

The `GetContentStorageUnit` method returns a reference to the content storage unit for this link-source object.

```
ODStorageUnit GetContentStorageUnit (in ODLinkKey key);
```

`key`	A valid link key obtained by a prior call to the `Lock` method.
return value	A reference to this link-source object's content storage unit.

DISCUSSION

The `key` parameter ensures thread safety. Before calling this method, you must call the `Lock` method to obtain this key.

The returned storage unit remains valid until the key is relinquished by the `Unlock` method. The link-source object handles the creation and destruction of its content storage unit, so you must neither dispose of nor release the returned storage unit.

You must not cache the returned storage unit; instead you must call this method whenever you need to access the content storage unit.

EXCEPTIONS

`kODErrInvalidLinkKey`	The `key` parameter is not a valid key for this link-source object.

SEE ALSO

The `ODLinkKey` type (page 894).
The `ODLinkSource::Lock` method (page 372).
The `ODLinkSource::Unlock` method (page 377).
The `ODStorageUnit` class (page 641).

GetUpdateID

The `GetUpdateID` method returns the update ID that uniquely identifies the current generation or version of this link-source object's content.

```
ODUpdateID GetUpdateID ();
```

return value The update ID of this link-source object's content.

DISCUSSION

If your part is the source part for this link-source object, you can call this method to determine the current version of this link-source object's content.

You can compare the returned update ID with a previously saved ID to establish whether the content needs to be updated. There is no implicit ordering of update ID values; they offer tests for equality only, with no indication of the time or nature of any link changes.

SEE ALSO

The `ODUpdateID` type (page 887).
The `ODLinkSource::GetChangeTime` method (page 369).
The `ODLinkSource::GetUpdateID` method (page 371).

IsAutoUpdate

The `IsAutoUpdate` method returns a Boolean value that indicates whether this link-source object's update mode is automatic.

```
ODBoolean IsAutoUpdate();
```

return value `kODTrue` if the update mode for this link-source object is automatic, otherwise `kODFalse`.

DISCUSSION

If your part is the source part of this link-source object, you can call this method to determine when your part should update the content of this link-source object.

If this method returns true, your part should update this link-source object whenever the source content is changed. If it returns false, your part should update this link-source object when the user clicks the Update Now button in the Link Source Info dialog box. (Whenever your part updates this link-source object, it should also call the `ContentUpdated` method.)

The update mode can be changed by the `SetAutoUpdate` method.

SEE ALSO

The `ODLinkSource::ContentUpdated` method (page 368).
The `ODLinkSource::SetAutoUpdate` method (page 373).
The `ODLinkSource::ShowLinkSourceInfo` method (page 375).

Lock

The `Lock` method locks this link-source object, ensuring exclusive access to its content storage unit.

```
ODBoolean Lock (in ODULong wait,
                out ODLinkKey key);
```

`wait`	The time interval to wait for access to be granted.
`key`	If access is granted, a valid link key; otherwise an undefined, invalid key.
return value	`kODTrue` if access is granted, otherwise `kODFalse`.

DISCUSSION

To ensure thread-safe access, you must call this method to acquire a valid link key before you write the link data. This method grants exclusive access to this link-source object's content; nested calls to the `Lock` method deny access.

The `wait` parameter specifies the time you are willing to wait for access to be granted. A value of 0 means no wait and is the only value accepted on the Mac OS platform. Other platforms may accept other values with platform-dependent meanings.

While your part has this link-source object locked, you must pass the key returned in the `key` output parameter to all methods that access or modify this link-source object, such as the `Clear`, `ContentUpdated`, and `GetContentStorageUnit` methods. When you are finished modifying this link-source object, you must pass the key to the `Unlock` method to unlock the link-source object.

EXCEPTIONS

`kODErrBrokenLink`	Internal error; this link-source object disconnected from its destinations.

SEE ALSO

The `ODLinkKey` type (page 894).
The `ODLinkSource::Clear` method (page 366).
The `ODLinkSource::ContentUpdated` method (page 368).
The `ODLinkSource::GetContentStorageUnit` method (page 370).
The `ODLinkSource::Unlock` method (page 377).

SetAutoUpdate

The `SetAutoUpdate` method sets this link-source object's update mode to automatic or manual.

```
void SetAutoUpdate (in ODBoolean automatic);
```

`automatic` `kODTrue` to set the update mode for this link-source object to automatic and `kODFalse` to set it to manual.

DISCUSSION

If your part is the source part of this link-source object, after you call the `ShowLinkSourceInfo` method, you can call this method to set the update mode to the mode the user selected in the Link Source Info dialog box. You can also call this method if you want to implement your own link information dialog boxes.

Your part should use the new update mode to determine when to update the content of this link-source object. If the `automatic` parameter is true, your part should update this link-source object whenever the source content is changed. If the parameter is false, your part should update this link-source object when the user clicks the Update Now button in the Link Source Info dialog box. (Whenever your part updates this link-source object, it should also call the `ContentUpdated` method.)

SEE ALSO

The `ODLinkSource::ContentUpdated` method (page 368).
The `ODLinkSource::IsAutoUpdate` method (page 371).
The `ODLinkSource::ShowLinkSourceInfo` method (page 375).

SetSourcePart

The `SetSourcePart` method establishes the source part that is to maintain this link-source object.

```
void SetSourcePart (in ODStorageUnit sourcePartSU);
```

`sourcePartSU`
A reference to the storage unit for the new source part.

DISCUSSION

Both parts and container applications call this method, typically when this link-source object is cloned during data-transfer operations involving the linked content.

If the user cuts content from your part that includes the link source, your part relinquishes ownership of this link-source object. If the user subsequently undoes that action, you must call this method to reclaim ownership.

After this method executes successfully, the reference count of the storage unit of the previous source part is decremented and the reference count of the new source part's storage unit is incremented.

ShowLinkSourceInfo

Mac OS

The `ShowLinkSourceInfo` method displays the Link Source Info dialog box for this link-source object.

```
ODBoolean ShowLinkSourceInfo (
                        in ODFacet facet,
                        in ODUpdateID change,
                        in ODBoolean changesAllowed,
                        out ODLinkInfoResult infoResult);
```

`facet` A reference to the facet displaying the selected link source.

`change` The update ID of the source part's source content.

`changesAllowed`
`kODTrue` if the user may change characteristics of this link-source object, otherwise `kODFalse`.

`infoResult`
A structure reflecting the user's selections in the Link Source Info dialog box.

return value `kODTrue` if the user did not cancel the Link Source Info dialog box, otherwise `kODFalse`.

DISCUSSION

You call this method in your part's `HandleEvent` method when the user selects the Link Info item from the Edit menu and the current selection is the border of a link source.

If the `change` parameter is different from this link-source object's current update ID (as returned by the `GetUpdateID` method), this link-source object does not have the most recent source content. This situation should occur only if the update mode is manual. In that case, the Update Now button is enabled to allow the user to request the link source to be updated from the source part.

Part viewers (as opposed to part editors) should pass false for the `changesAllowed` parameter. Changes are disabled automatically if the draft is read only.

The Link Source Info dialog box displays the part kind of the linked data, together with the dates and times of its creation and last update. The dialog box lets the user set the link update mode to automatic (On Save) or manual. The user can also break the link or request an immediate update from the source part to this link-source object.

If the user exits the Link Source Info dialog box by clicking a button other than Cancel, this method returns true and you should examine the `action` field of the `infoResult` output parameter to determine what action to take in response to the user's selections.

- If the `action` field is `kODLinkInfoBreakLink`, your part should relinquish ownership of this link-source object. This operation should be undoable.
- If the `action` field is `kODLinkInfoUpdateNow`, you should update this link-source object from your part's content and call the `ContentUpdated` method. This action should not be undoable in view of the part's difficulty in restoring the previous link content.
- If the `action` field is `kODLinkInfoOk`, you should examine the `autoUpdate` field of the `infoResult` output parameter. If the `autoUpdate` field indicates a change to the update mode for this link-source object (as obtained by calling the `IsAutoUpdate` method), you should call the `SetAutoUpdate` method to change the update mode. This operation should not be undoable.

  If the user has changed from manual mode to automatic, you should call the `GetUpdateID` method to see whether this link-source object has the most

up-to-date version of your part's source content. If not, you should update the link-source object and call the `ContentUpdated` method.

If the user cancels the Link Source Info dialog box, this method returns false and you do not need to take any further action.

EXCEPTIONS

`kODErrBrokenLink`	Internal error; this link-source object disconnected from its destinations.
`kODErrNullFacetInput`	The `facet` parameter is null.
`kODErrNullLinkInfoResultInput`	The `infoResult` parameter is null.

SEE ALSO

The `ODLinkInfoAction` type (page 893).
The `ODLinkInfoResult` type (page 894).
The `ODUpdateID` type (page 887).
The `ODLink::ShowLinkDestinationInfo` method (page 347).
The `ODLinkSource::ContentUpdated` method (page 368).
The `ODLinkSource::GetUpdateID` method (page 371).
The `ODLinkSource::IsAutoUpdate` method (page 371).
The `ODLinkSource::SetAutoUpdate` method (page 373).

Unlock

The `Unlock` method unlocks this link-source object, relinquishing access to its content storage unit.

```
void Unlock (in ODLinkKey key);
```

`key` A valid key obtained by a prior call to the `Lock` method.

DISCUSSION

You should call this method as soon as possible after accessing or modifying the content storage unit of this link-source object.

The `key` parameter should be a valid key obtained by an earlier call to the `Lock` method. After this method executes successfully, the specified key is no longer valid and access to the link-source object's content is relinquished.

EXCEPTIONS

`kODErrInvalidLinkKey`	The `key` parameter is not a valid key for this link-source object.

SEE ALSO

The `ODLinkKey` type (page 894).
The `ODLinkSource::Lock` method (page 372).

ODLinkSpec

Superclasses `ODObject`

Subclasses none

An object of the `ODLinkSpec` class advertises a part's ability to create a link to data it is transferring.

Description

A link specification is a signal that a link can be made from content in a specified source part. The link specification remains valid as long as its source part exists and the document containing it remains opened. After the document is closed, the link specification becomes meaningless.

When a source part that supports linking writes to the clipboard or drag-and-drop object, it uses a link specification to advertise its ability to create a link. The source part creates a link specification by calling its draft's `CreateLinkSpec` method (page 164), passing as parameters a reference to itself and data sufficient to enable the source part to identify the selected content. The data in a link specification is private to the source part; it is returned to the source part if the part's `CreateLink` method (page 476) is called to create a link. Because the link specification is valid only during the lifetime of its source part, the data can contain pointers to information maintained by the part.

A link specification provides methods for writing or reading itself to and from a focused storage unit. In addition to writing content to the storage content unit of the data-transfer object (clipboard or drag-and-drop object), the source part calls the link specification's `WriteLinkSpec` method (page 382) to write the link specification to the `kODPropLinkSpec` property.

A user who pastes or drops the source data to a destination part can request a link by means of the Paste As dialog box. The destination part creates an empty link specification by calling its draft's `CreateLinkSpec` method, passing null for the source part and data parameters. The destination part calls the

`ReadLinkSpec` method (page 381) of the empty link specification to read from the `kODPropLinkSpec` property of the content storage unit. The destination part then creates the link by passing the link specification to its draft's `AcquireLink` method (page 152), which in turn passes the link specification back to the source part's `CreateLink` method.

Whenever a source part writes a link specification to the clipboard, it should save clipboard's current update ID, as returned by the clipboard's `GetUpdateID` method (page 91). The part must remove the link specification from the clipboard when either of the following conditions is true:

- It becomes infeasible to create the link, for example, because the potential source content is deleted or modified.
- The part's `ReleaseAll` method (page 454) is called.

The source part can check the clipboard update ID and compare it with the saved ID to determine whether its content is still on the clipboard.

For information on using the clipboard and drag-and-drop object, see the descriptions of the classes `ODClipboard` (page 81) and `ODDragAndDrop` (page 184). For further information on the implementation of OpenDoc links, see the descriptions of the companion classes `ODLink` (page 339) and `ODLinkSource` (page 361), and the chapter on data transfer in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODLinkSpec` methods, followed by detailed descriptions in alphabetical order.

`ReadLinkSpec`	Initializes this empty link specification by reading its data from the specified focused storage unit.
`WriteLinkSpec`	Writes the data for this link specification into the specified focused storage unit.

ReadLinkSpec

The `ReadLinkSpec` method initializes this empty link specification by reading its data from the specified focused storage unit.

```
void ReadLinkSpec (in ODStorageUnit su);
```

`su` A reference to the storage unit whose focused value contains the link-specification data.

DISCUSSION

If your part is a destination part, you create an empty link specification by calling the draft's `CreateLinkSpec` method, passing null for the source part and data parameters. Then, call the empty link specification's `ReadLinkSpec` method to initialize the link specification from data in the content storage unit of the clipboard or drag-and-drop object. The storage unit should be focused on the value of type `kODLinkSpec` in the `kODPropLinkSpec` property.

EXCEPTIONS

`kODErrCorruptLinkSpecValue`
The focused storage unit contains an invalid link-specification value.

`kODErrNoLinkSpecValue`
The focused property does not contain a link-specification value.

`kODErrOutOfMemory` There is not enough memory to read the link specification.

`kODErrUnknownLinkSpecVersion`
The link-specification version is not recognized.

SEE ALSO

The `ODDraft::CreateLinkSpec method (page 164).`
The `ODLinkSpec::WriteLinkSpec` method (page 382).

WriteLinkSpec

The `WriteLinkSpec` method writes the data for this link specification into the specified focused storage unit.

```
void WriteLinkSpec (in ODStorageUnit su);
```

`su`	A reference to the storage unit where the link-specification data is to be written.

DISCUSSION

If your part is a source part that supports linking, you call this method to write a link specification to the content storage unit of the clipboard or drag-and-drop object. The storage unit should be focused on the `kODPropLinkSpec` property. This method writes the link specification to the value of type `kODLinkSpec` in the focused property, replacing any link specification that was previously stored in that value or creating the value if it doesn't already exist.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to write the link specification.

SEE ALSO

The `ODLinkSpec::ReadLinkSpec` method (page 381).

ODMenuBar

Superclasses `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODMenuBar` class represents a composite menu bar object, made up of menus from the document shell and the active part.

Description

When OpenDoc first opens a document, the document shell creates a menu bar object, adds menus to it, and installs it as the base menu bar object, which contains the default set of menus shared by all parts in the document. Part editors can obtain a copy of the base menu bar object by calling the window-state object's `CopyBaseMenuBar` method (page 826), add menus to it, and install it as the current menu bar object .

Your part creates a empty menu bar by calling the window-state object's `CreateMenuBar` method (page 829). Your part can also create a copy of an existing menu bar object by calling its menu bar object's `Copy` method (page 388). These methods return a reference to a menu bar object.

OpenDoc allows the document shell and part editors to register command IDs for menu items. If no command ID is registered, a **synthetic** command ID, one that is manufactured from the menu and menu item IDs, is generated. If a command ID is not registered and not synthetic, a part should not handle it.

Methods

This section presents summary descriptions of the `ODMenuBar` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [M] are specific to the Mac OS platform.

Menu Bars

`Copy`	Copies this menu bar object.
`Display`	Installs this menu bar object as the current menu bar, making it visible and active.
`IsValid`	Returns a Boolean value that indicates whether this menu bar object is equivalent to the base menu bar object from which it was copied.

Menus

`AddMenuBefore`	Inserts a new menu before another specified menu on this menu bar.
`AddMenuLast`	Appends a new menu to the end of this menu bar.
`AddSubMenu` [M]	Adds a submenu to this menu bar.
`DisableAll` [M]	Disables all menus in this menu bar, except for system menus (for example, the Apple menu and the Guide menu).
`EnableAll` [M]	Enables all menus in this menu bar, except for system menus (for example, the Apple menu and the Guide menu).
`GetMenu`	Returns a platform-specific menu structure for the specified menu ID.
`RemoveMenu`	Removes the menu with the specified ID from this menu bar.

Menu Items

`GetItemString` [M]	Gets the text string of the specified menu item.
`SetItemString` [M]	Sets the text string for the specified menu item.
`EnableCommand` [M]	Enables or disables the specified menu item.
`CheckCommand` [M]	Places a checkmark next to the specified menu item, or unchecks it.
`EnableAndCheckCommand` [M]	Enables or disables, and checks or unchecks, the specified menu item.
`GetMenuAndItem` [M]	Gets the menu/menu item designation for the specified command ID.

`GetCommand` [M] Returns the command ID for the specified menu/menu item designation.

`RegisterCommand` [M] Associates a command ID with the specified menu/menu item designation.

`UnregisterCommand` [M] Removes the association between the specified command ID and its menu/menu item designation.

`UnregisterAll` [M] Unregisters all command IDs currently registered with this menu bar object.

Menu Commands

`IsCommandRegistered` [M] Returns a Boolean value that indicates whether the specified command ID is registered with this menu bar object.

`IsCommandSynthetic` [M] Returns a Boolean value that indicates whether the specified command ID is synthetic.

AddMenuBefore

The `AddMenuBefore` method inserts a new menu before another specified menu on this menu bar.

```
void AddMenuBefore (in ODMenuID menuID,
                    in ODPlatformMenu menu,
                    in ODPart part,
                    in ODMenuID beforeID);
```

`menuID` A platform-specific identifier of the new menu. On the Mac OS platform, the menu ID is defined as a signed 16-bit value.

`menu` A 32-bit value identifying the platform-specific menu for the specified menu ID. On the Mac OS platform, this parameter is a menu handle (type `MenuHandle`).

`part`	A reference to a part that owns the menu, or `kODNULL` if the menu is a document shell menu.
`beforeID`	The menu ID of the menu that follows the new menu.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to allocate the menu.

AddMenuLast

The `AddMenuLast` method appends a new menu to the end of this menu bar.

```
void AddMenuLast (in ODMenuID menuID,
                  in ODPlatformMenu menu,
                  in ODPart part);
```

`menuID`	A platform-specific identifier of the new menu. On the Mac OS platform, the menu ID is defined as a signed 16-bit value.
`menu`	A 32-bit value identifying the platform-specific menu for the specified menu ID. On the Mac OS platform, this parameter is a menu handle (type `MenuHandle`).
`part`	A reference to a part that owns the menu, or `kODNULL` if the menu is a document shell menu.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to allocate the menu.

AddSubMenu

Mac OS

The `AddSubMenu` method adds a submenu to this menu bar.

```
void AddSubMenu (in ODMenuID menuID,
                 in ODPlatformMenu menu,
                 in ODPart part);
```

`menuID`	A platform-specific identifier of the new submenu. On the Mac OS platform, the menu ID is defined as a signed 16-bit value.
`menu`	A 32-bit value identifying the platform-specific menu for the specified menu ID. On the Mac OS platform, this parameter is a menu handle (type `MenuHandle`).
`part`	A reference to a part that owns the submenu, or `kODNULL` if the submenu is a document shell submenu.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to allocate the submenu.

CheckCommand

Mac OS

The `CheckCommand` method places a checkmark next to the specified menu item, or unchecks it.

```
void CheckCommand (in ODCommandID cmdNumber,
                   in ODBoolean check);
```

`cmdNumber`	The command ID of the menu item to be checked. On the Mac OS platform, the command ID is defined as a signed 32-bit value.
`check`	`kODTrue` if the menu item is to be checked, or `kODFalse` if the menu item is to be unchecked.

SEE ALSO

The `ODCommandID` type (page 865).

Copy

The `Copy` method copies this menu bar object.

```
ODMenuBar Copy ();
```

return value A reference to a menu bar object to be copied.

DISCUSSION

OpenDoc calls this method; this method is not called by most parts.

DisableAll

Mac OS

The `DisableAll` method disables all menus in this menu bar, except for system menus (for example, the Apple menu and the Guide menu).

```
void DisableAll ();
```

SEE ALSO

The `ODMenuBar::EnableAll` method (page 389).

Display

The `Display` method installs this menu bar object as the current menu bar, making it visible and active.

```
void Display ();
```

EnableAll

Mac OS

The `EnableAll` method enables all menus in this menu bar, except for system menus (for example, the Apple menu and the Guide menu).

```
void EnableAll ();
```

SEE ALSO

The `ODMenuBar::DisableAll` method (page 388).

EnableAndCheckCommand

Mac OS

The `EnableAndCheckCommand` method enables or disables, and checks or unchecks, the specified menu item.

```
void EnableAndCheckCommand (in ODCommandID cmdNumber,
                            in ODBoolean enable,
                            in ODBoolean check);
```

`cmdNumber` The command ID of the menu item to be enabled or disabled, or checked or unchecked. On the Mac OS platform, the command ID is defined as a signed 32-bit value.

`enable`	`kODTrue` if the menu item is to be enabled, or `kODFalse` if the menu item is to be disabled.
`check`	`kODTrue` if the menu item is to be checked, or `kODFalse` if the menu item is to be unchecked.

SEE ALSO

The `ODCommandID` type (page 865).

EnableCommand

Mac OS

The `EnableCommand` method enables or disables the specified menu item.

```
void EnableCommand (in ODCommandID cmdNumber,
                    in ODBoolean enable);
```

`cmdNumber`	The command ID of the menu item to be enabled or disabled. On the Mac OS platform, the command ID is defined as a signed 32-bit value.
`enable`	`kODTrue` if the menu item is to be enabled, or `kODFalse` if the menu item is to be disabled.

SEE ALSO

The `ODCommandID` type (page 865).

GetCommand

Mac OS

The `GetCommand` method returns the command ID for the specified menu/menu item designation.

```
ODCommandID GetCommand (in ODMenuID menu,
                        in ODMenuItemID menuItem);
```

`menu`	A platform-specific identifier for a menu. On the Mac OS platform, the menu ID is defined as a signed 16-bit value.
`menuItem`	A platform-specific identifier for a menu item. On the Mac OS platform, the menu item ID is defined as a signed 16-bit value.
return value	The command ID for the specified menu/menu item designation, or `kODNoCommand` if the command ID does not exist. On the Mac OS platform, the command ID is defined as a signed 32-bit value.

SEE ALSO

The `ODCommandID` type (page 865).

GetItemString

Mac OS

The `GetItemString` method gets the text string of the specified menu item.

```
void GetItemString (in ODCommandID cmdNumber,
                    out ODIText itemString);
```

`cmdNumber`	The command ID of the menu item whose text you wish to obtain. On the Mac OS platform, the command ID is defined as a signed 32-bit value.
`itemString`	The text string of the specified menu item.

EXCEPTIONS

`kODErrInvalidCommandID`
The specified menu item does not exist.

SEE ALSO

The `ODCommandID` type (page 865).
The `ODIText` type (page 845).

GetMenu

The `GetMenu` method returns a platform-specific menu structure for the specified menu ID.

```
ODPlatformMenu GetMenu (in ODMenuID menu);
```

`menu`	A platform-specific identifier for a menu. On the Mac OS platform, the menu ID is defined as a signed 16-bit value.
return value	A 32-bit value identifying the platform-specific menu for the specified menu ID. On the Mac OS platform, the return value is a menu handle (type `MenuHandle`).

GetMenuAndItem

Mac OS

The `GetMenuAndItem` method gets the menu/menu item designation for the specified command ID.

```
void GetMenuAndItem (in ODCommandID command,
                     out ODMenuID menu,
                     out ODMenuItemID menuItem);
```

`command`	The command ID of the menu item to be checked. On the Mac OS platform, the command ID is defined as a signed 32-bit value.
`menu`	A platform-specific identifier for a menu. On the Mac OS platform, the menu ID is defined as a signed 16-bit value.
`menuItem`	A platform-specific identifier for a menu item. On the Mac OS platform, the menu item ID is defined as a signed 16-bit value.

EXCEPTIONS

`kODErrInvalidCommandID`
The specified menu item does not exist.

SEE ALSO

The `ODCommandID` type (page 865).

IsCommandRegistered

Mac OS

The `IsCommandRegistered` method returns a Boolean value that indicates whether the specified command ID is registered with this menu bar object.

```
ODBoolean IsCommandRegistered (in ODCommandID command);
```

`command`	The command ID of the menu item to be checked. On the Mac OS platform, the command ID is defined as a signed 32-bit value.
return value	`kODTrue` if the specified command ID is registered with this menu bar object, otherwise `kODFalse`.

SEE ALSO

The `ODCommandID` type (page 865).

IsCommandSynthetic

Mac OS

The `IsCommandSynthetic` method returns a Boolean value that indicates whether the specified command ID is synthetic.

```
ODBoolean IsCommandSynthetic (in ODCommandID command);
```

`command` The command ID of the menu item to be checked. On the Mac OS platform, the command ID is defined as a signed 32-bit value.

return value `kODTrue` if the specified command ID is synthetic, otherwise `kODFalse`.

DISCUSSION

A synthetic command ID is one that is not registered with this menu bar object, but that was instead manufactured from the menu and menu item IDs.

SEE ALSO

The `ODCommandID` type (page 865).
The `ODMenuBar::RegisterCommand` method (page 395).

IsValid

The `IsValid` method returns a Boolean value that indicates whether this menu bar object is equivalent to the base menu bar object from which it was copied.

```
ODBoolean IsValid ();
```

return value `kODTrue` if this menu bar object is equivalent to the base menu bar object from which it was copied, otherwise `kODFalse`.

DISCUSSION

Your part calls its cached base menu bar's `IsValid` method before adding its own menus and displaying the composite menu bar. If this base menu bar is no longer valid, your part recopies the current base menu bar object by calling its window-state's `CopyBaseMenuBar` method.

SEE ALSO

The `ODWindowState::CopyBaseMenuBar` method (page 826).

RegisterCommand

Mac OS

The `RegisterCommand` method associates a command ID with the specified menu/menu item designation.

```
void RegisterCommand (in ODCommandID command,
                      in ODMenuID menu,
                      in ODMenuItemID menuItem);
```

`command`	The command ID to be associated with the specified menu/menu item designation. On the Mac OS platform, the command ID is defined as a signed 32-bit value.
`menu`	A platform-specific identifier for a menu. On the Mac OS platform, the menu ID is defined as a signed 16-bit value.
`menuItem`	A platform-specific identifier for a menu item. On the Mac OS platform, the menu item ID is defined as a signed 16-bit value.

DISCUSSION

Command IDs above 20,000 are reserved for parts. If your part registers command IDs, it should use numbers greater than 20,000.

SEE ALSO

The `ODCommandID` type (page 865).
The `ODMenuBar::UnregisterCommand` method (page 397).

RemoveMenu

The `RemoveMenu` method removes the menu with the specified ID from this menu bar.

```
void RemoveMenu (in ODMenuID menu);
```

`menu` A platform-specific identifier of the menu to be removed. On the Mac OS platform, the menu ID is defined as a signed 16-bit value.

DISCUSSION

The displayed menu bar is not affected.

SetItemString

Mac OS

The `SetItemString` method sets the text string for the specified menu item.

```
void SetItemString (in ODCommandID cmdNumber,
                    in ODIText itemString);
```

`cmdNumber` The command ID of the menu item to be changed. On the Mac OS platform, the command ID is defined as a signed 32-bit value.

`itemString` The desired text string for the specified menu item.

SEE ALSO

The `ODCommandID` type (page 865).
The `ODIText` type (page 845).

UnregisterAll

Mac OS

The `UnregisterAll` method unregisters all command IDs currently registered with this menu bar object.

```
void UnregisterAll ();
```

UnregisterCommand

Mac OS

The `UnregisterCommand` method removes the association between the specified command ID and its menu/menu item designation.

```
void UnregisterCommand (in ODCommandID command);
```

`command` The command ID to be unregistered. On the Mac OS platform, the command ID is defined as a signed 32-bit value.

SEE ALSO

The `ODCommandID` type (page 865).
The `ODMenuBar::RegisterCommand` method (page 395).

ODMessageInterface

Superclasses `ODObject`

Subclasses none

An object of the `ODMessageInterface` class provides the capability of creating and sending semantic events. This class works with the `ODSemanticInterface` class to handle messaging between parts.

Description

When a document is opened, the session object creates a single message interface object. All parts of that document share the same message interface object; you can obtain a reference to it by calling the session object's `GetMessageInterface` method (page 585).

The OpenDoc message interface object is responsible for constructing Apple events, getting and setting event attributes, parsing events, and sending events. OpenDoc supports your part editor's ability to create and send semantic events through another object, the message interface, and to other parts. Your part can also send semantic events to the document shell by specifying the constant `kODAppShell` for the destination part. The message interface is also the object through which OpenDoc sends semantic events to your part, although your part does not make any calls to the message interface in that situation.

For more information related to semantic events, see the `ODSemanticInterface` class description (page 557). For more information related to creating and sending Apple events, see the "Creating and Sending Apple Events" chapter of *Inside Macintosh: Interapplication Communication*. For general information on scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODMessageInterface` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Creating Events

`CreateEvent`	Creates an Apple event object.
`CreatePartAddrDesc`	Creates an address descriptor that specifies the address of the specified part.
`CreatePartObjSpec`	Creates an object specifier that refers to the specified part.

Sending Events

`ProcessSemanticEvent`	Dispatches an Apple event object to the appropriate handler.
`Send`	Sends the specified Apple event object and requests a reply if appropriate.

CreateEvent

The `CreateEvent` method creates an Apple event object.

```
ODSShort CreateEvent (in ODEventClass theAEEventClass,
                      in ODEventID theAEEventID,
                      in ODAddressDesc target,
                      in ODSLong transactionID,
                      out ODAppleEvent theResult);
```

`theAEEventClass`
The event class of the Apple event object to be created.

`theAEEventID`
The event ID of the Apple event object to be created.

`target` A reference to the address of the destination part.

`transactionID`
: A value that uniquely identifies this transaction.

`theResult`
: A reference to the new Apple event object.

return value
: The return ID assigned to this event.

DISCUSSION

Your part calls this method instead of the `AECreateAppleEvent` method defined by the Apple Event Manager. Your part does not need to specify a return ID for the Apple event object. Instead, OpenDoc assigns a return ID and uses that value to keep track of the sender of the event. The `transactionID` parameter should be used by your part to identify a related group of Apple event objects and their corresponding replies. If you are creating an event to send to another part, use the message interface's `CreatePartAddrDesc` method to create the `target` parameter.

After this method executes successfully, your part may add whatever data it wants to the resulting Apple event object and then send it using the message interface's `Send` method.

EXCEPTIONS

`kODErrOutOfMemory`
: There is not enough memory to allocate the event object.

The Apple Event Manager may throw other exceptions.

SEE ALSO

The `ODEventClass` type (page 895).
The `ODEventID` type (page 895).
The `ODMessageInterface::CreatePartAddrDesc` method (page 401).
The `ODMessageInterface::Send` method (page 403).
The `ODAddressDesc` class (page 39).
The `ODAppleEvent` class (page 41).

CreatePartAddrDesc

The `CreatePartAddrDesc` method creates an address descriptor that specifies the address of the specified part.

```
void CreatePartAddrDesc (
                        out ODAddressDesc theAddressDesc,
                        in ODPart part);
```

`theAddressDesc`
A reference to the address descriptor object to be set to the address of the specified part.

`part` A reference to the part whose address is desired.

DISCUSSION

You call this method to create an address descriptor that identifies the OpenDoc document in which the destination part resides.

The address descriptor returned by this method should not be stored persistently because it is only valid while the part is instantiated.

EXCEPTIONS

This method may throw platform-specific exceptions.

SEE ALSO

The `ODMessageInterface::CreatePartObjSpec` method (page 402).
The `ODAddressDesc` class (page 39).

CreatePartObjSpec

The `CreatePartObjSpec` method creates an object specifier that refers to the specified part.

```
void CreatePartObjSpec (out ODObjectSpec theObjSpec,
                        in ODPart thePart);
```

`theObjSpec` A reference to the object specifier to be set to refer to the specified part.

`thePart` A reference to the part for which an object specifier is created.

DISCUSSION

If your part sends a semantic event to another part, you may use this method to address that part or to create an object specifier that represents that part. You must also call your message interface object's `CreatePartAddrDesc` method to create an address descriptor that identifies the OpenDoc document in which the destination part resides.

The object specifier created by this method has the type `cPart` and has a null container type. The object specifier allows your parts to communicate privately with each other and should not be stored persistently because it is only valid while the part is instantiated. The object specifier should also not be used in a recordable event.

EXCEPTIONS

This method may throw platform-specific exceptions.

SEE ALSO

The `ODMessageInterface::CreatePartAddrDesc` method (page 401).
The `ODObjectSpec` class (page 440).

ProcessSemanticEvent

The `ProcessSemanticEvent` method dispatches an Apple event object to the appropriate handler.

```
ODBoolean ProcessSemanticEvent (in ODEventData theEvent);
```

`theEvent`	A platform-specific structure representing an event. On the Mac OS platform, the structure is defined as a Mac OS event record.
return value	`kODTrue` if the event was processed, otherwise `kODFalse`.

DISCUSSION

OpenDoc calls this method to dispatch Apple event objects to the appropriate part. Your part should not call this method.

SEE ALSO

The `ODEventData` type (page 860).

Send

The `Send` method sends the specified Apple event object and requests a reply if appropriate.

```
void Send (in ODFrame toFrame,
           in ODPart fromPart,
           in ODAppleEvent theAppleEvent,
           in ODAppleEvent reply,
           in ODSendMode sendMode,
           in ODSendPriority sendPriority,
           in ODULong timeOutInTicks);
```

`toFrame` A reference to a frame belonging to the part sending the Apple event object, or `kODNULL` if the part sending the Apple event object has no frame.

`fromPart` A reference to the part sending the Apple event object.

`theAppleEvent`
A reference to an Apple event object to send.

`reply` A reference to a reply Apple event object to be returned.

`sendMode` The flags that specify the interactions between the sending and receiving parts.

`sendPriority`
The priority of the event.

`timeOutInTicks`
The number of ticks to wait for a reply before generating a time-out exception, expressed in ticks (60ths of a second).

DISCUSSION

The Apple Event Manager defines constants that can be used for the `sendMode` and `sendPriority` parameters. These constants are described in the "Creating and Sending Apple Events" chapter of *Inside Macintosh: Interapplication Communication*.

Before calling this method, you must create the Apple event object to send using the message interface's `CreateEvent` method. Your part is responsible for deleting both the Apple event object and any reply that was returned.

EXCEPTIONS

This method may throw platform-specific exceptions.

SEE ALSO

The `ODSendMode` type (page 895).
The `ODSendPriority` type (page 895).
The `ODMessageInterface::CreateEvent` method (page 399).
The `ODMessageInterface::CreatePartAddrDesc` method (page 401).
The `ODAppleEvent` class (page 41).

ODNameResolver

Superclasses `ODObject`

Subclasses none

An object of the `ODNameResolver` class provides for resolution of object specifiers in semantic events. The `ODNameResolver` class works with the `ODSemanticInterface` class to resolve Apple event object specifiers sent between parts.

Description

When a document is opened, the session object creates a single name-resolver object. All parts of that document share the name resolver; you can obtain a reference to it by calling the session object's `GetNameResolver` method (page 586). Your part's semantic interface can use the name resolver to initiate the resolution of an object specifier and to call your part's object accessors. Your part can also use the name resolver to create a swap token, which allows your part to pass the resolution of an object specifier to another part.

For more information related to object accessors, see the `ODSemanticInterface` class description (page 557). For more information related to object specifiers and the Apple event object model, see the "Resolving and Creating Object Specifier Records" chapter of *Inside Macintosh: Interapplication Communication* and the chapter on scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODNameResolver` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Object Resolution

`CallObjectAccessor`	Calls the object accessor of the calling part's semantic interface.
`Resolve`	Resolves an object specifier into a token that identifies the target object.

Token Manipulation

`CreateSwapToken`	Creates a token that passes resolution of an object specifier to another part.
`DisposeToken`	Deallocates the internal structures of the specified token.
`GetContextFromToken`	Gets references to the part and the frame in whose context the specified token was created.
`IsODToken`	Returns a Boolean value that indicates whether the specified token object has been initialized to the proper format for an OpenDoc token.
`GetUserToken`	Returns a reference to the descriptor object contained in the specified OpenDoc token.

CallObjectAccessor

The `CallObjectAccessor` method calls the object accessor of the calling part's semantic interface.

```
void CallObjectAccessor (in ODPart part,
                         in ODDescType desiredClass,
                         in ODOSLToken containerToken,
                         in ODDescType containerClass,
                         in ODDescType keyForm,
                         in ODDesc keyData,
                         in ODOSLToken token);
```

`part` A reference to the part calling this method (that is, the context for the supplied token).

`desiredClass`
: The class of the desired Apple event object(s).

`containerToken`
: A reference to the token that identifies the container for the desired objects.

`containerClass`
: The class of the container for the desired Apple event object(s).

`keyForm`
: The key form specified by the object specifier record for the object(s) to be located.

`keyData`
: A reference to the descriptor object containing the key data specified by the object specifier record for the object(s) to be located.

`token`
: A reference to the token to be filled in by the object accessor being called.

DISCUSSION

You can use this method to call the object accessor for a particular object class in a particular container. This is especially useful for constructing a list of tokens. Your part must call this method instead of directly calling the `CallObjectAccessor` method of your part's semantic-interface object.

The `part` parameter must be a reference to the part that calls this method.

EXCEPTIONS

The Apple Event Manager may throw an exception if the object accessor could not be found for the specified part.

Parts and OpenDoc may throw other exceptions.

SEE ALSO

The `ODDescType` type (page 895).
The `ODSemanticInterface::CallObjectAccessor` method (page 572).
The "Resolving and Creating Object Specifier Records" chapter of *Inside Macintosh: Interapplication Communication*.

CreateSwapToken

The `CreateSwapToken` method creates a token that passes resolution of an object specifier to another part.

```
void CreateSwapToken (in ODOSLToken token,
                      in ODPart part,
                      in ODFrame frame);
```

`token`	A reference to the token to contain the part and frame information.
`part`	A reference to the part to be specified by the token.
`frame`	A reference to the frame to be specified by the token, or `kODNULL` if the specified part has no frames.

DISCUSSION

This method fills in the empty token you provide with information about the specified part and frame.

This method is intended to be called by your part's object accessors if they cannot handle the resolution of an object specifier, but know of another part that can (for instance, an embedded part). In this case, your object accessor can return a swap token to OpenDoc, which then tries to resolve the object specifier using the new part and frame.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to create the token data structures.

SEE ALSO

The `ODNameResolver::GetContextFromToken` method (page 409).
The `ODSemanticInterface::CallObjectAccessor` method (page 572).

DisposeToken

The `DisposeToken` method deallocates the internal structures of the specified token.

```
void DisposeToken (in ODOSLToken theToken);
```

`theToken` A reference to the token to be deallocated.

DISCUSSION

This method calls the semantic interface of the part that created the token to dispose of the token's internal data structures. If the semantic interface's `CallDisposeTokenProc` method does not handle the event, this method handles the disposal of the token.

EXCEPTIONS

This method may throw platform-specific exceptions.

SEE ALSO

The `ODSemanticInterface::CallDisposeTokenProc` method (page 567).

GetContextFromToken

The `GetContextFromToken` method gets references to the part and the frame in whose context the specified token was created.

```
void GetContextFromToken(in ODOSLToken token,
                         out ODPart part,
                         out ODFrame frame);
```

`token` A reference to the token to be examined.

`part` A reference to the part representing the context for this token.

`frame` A reference to the frame representing the context for this token, or `kODNULL` if the specified part does not have any frames associated with it.

DISCUSSION

The `token` parameter must be a valid OpenDoc token; you can call the `IsODToken` method to check whether a token is valid.

This method does not increment the reference count of either the part or the frame in whose context the token was created.

SEE ALSO

The `ODNameResolver::CreateSwapToken` method (page 408).
The `ODNameResolver::IsODToken` method (page 411).

GetUserToken

The `GetUserToken` method returns a reference to the descriptor object contained in the specified OpenDoc token.

```
ODDesc GetUserToken(in ODOSLToken token);
```

`token` A reference to the OpenDoc token from which to extract the descriptor.

return value A reference to the descriptor object contained inside this token.

DISCUSSION

This method returns the descriptor that represents your private data. You must not dispose of this descriptor; OpenDoc will dispose of it for you.

The `token` parameter must be a valid OpenDoc token; you can call the `IsODToken` method to check whether a token is valid.

EXCEPTIONS

`kODErrNotAnODToken`	The specified token is not an OpenDoc token.

SEE ALSO

The `ODNameResolver::IsODToken` method (page 411).

IsODToken

The `IsODToken` method returns a Boolean value that indicates whether the specified token object has been initialized to the proper format for an OpenDoc token.

```
ODBoolean IsODToken(in ODOSLToken token);
```

`token`	A reference to the token to check.
return value	`kODTrue` if the token is a token object created by one of your object accessors, otherwise `kODFalse`.

DISCUSSION

You can call this method from the `CallCompareProc` method of your part's semantic interface. Either of the `obj1` and `obj2` parameters to that method may be either an `ODDesc` object that describes data or an `ODOSLToken` object created by one of your object accessors. To distinguish these two possibilities, you can pass each input parameter, in turn, to the `IsODToken` method. If this method returns true, the input parameter is a token object; if it returns false, the input parameter is a descriptor object.

SEE ALSO

The `ODSemanticInterface::CallCompareProc` method (page 565).

Resolve

The `Resolve` method resolves an object specifier into a token that identifies the target object.

```
void Resolve (in ODObjectSpec theObject,
              in ODOSLToken token,
              in ODPart contextPart);
```

`theObject` A reference to the object specifier to be resolved.

`token` A reference to the final token produced by the resolution.

`contextPart`
A reference to the part calling this method (that is, the context for the supplied token).

DISCUSSION

OpenDoc calls this method to resolve object specifiers in the direct parameter of an Apple event. In addition, your part should call this method to begin the resolution of an object specifier. Note that this method does not allow you to specify any callback flags. These flags are set by the semantic interface of each part using the semantic interface's `SetOSLSupportFlags` method. This method may call the object accessors of one or more parts to resolve the object specifier.

The `contextPart` parameter must be a reference to the part that calls this method.

Your part is responsible for deleting the returned token when it is no longer needed by calling the name resolver's `DisposeToken` method.

EXCEPTIONS

`kODErrOutOfMemory` There is not enough memory to allocate the needed internal structures.

The Apple Event Manager may throw an exception if the specified part was invalid or the part does not support the semantic interface extension.

Parts and OpenDoc may throw other exceptions.

SEE ALSO

The `ODNameResolver::CallObjectAccessor` method (page 406).
The `ODNameResolver::DisposeToken` method (page 409).
The `ODSemanticInterface::CallObjectAccessor` method (page 572).
The `ODSemanticInterface::SetOSLSupportFlags` method (page 577).
The chapter on scripting in the *OpenDoc Programmer's Guide for the Mac OS*.
The "Resolving and Creating Object Specifier Records" chapter of *Inside Macintosh: Interapplication Communication*.

ODNameSpace

Superclasses `ODObject`

Subclasses `ODObjectNameSpace`, `ODValueNameSpace`

An object of the `ODNameSpace` class associates an object or data structure with a unique key and provides a fast way to retrieve information.

Description

A name space object allows a part to identify an object or value using a unique key, which can be passed easily between parts. Parts can use name spaces to store references to parts that support, for example, scripting systems. A part can then iterate over the content of the name space to obtain a list of scriptable parts or to send a message to each of the scriptable parts. Name spaces can also be written to a storage unit and read back in later.

The `ODNameSpace` class is an abstract class that defines the basic name-space functionality. OpenDoc defines the subclasses corresponding to two types of name space: object and value. An object of the `ODObjectNameSpace` class (page 436) represents an object name space, which stores objects; an object of the `ODValueNameSpace` class (page 793) represents a value name space, which stores values of any types as byte arrays.

Your part creates a new name space by calling the name-space manager's `CreateNameSpace` method (page 422), specifying a unique name that OpenDoc can use to identify the new name space. Entries within the name space must similarly be identified by a unique key. To obtain a reference to an existing name space, your part calls the name-space manager's `HasNameSpace` method (page 424).

Name spaces can be arranged hierarchically to allow you to search multiple name spaces for a single key. Searches move from a child name space to its parent name space until the entry is found or until there are no more name spaces to search. A search also stops if the type of the parent name space is

different from the type of the child name space, for example, if a value name space is the child of an object name space.

Methods

This section presents summary descriptions of the `ODNameSpace` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Name-Space Entries

`Unregister`	Removes the specified entry from this name space.
`Exists`	Returns a Boolean value that indicates whether this name space contains an entry with the specified key.

Name-Space Attributes

`GetName`	Returns the unique name of this name space.
`GetParent`	Returns a reference to the parent name space of this name space.
`GetType`	Returns the type of this name space.
`SetType`	Sets the type of this name space.

Name-Space Storage

`ReadFromStorage`	Reads the content of this name space from the specified storage-unit view.
`WriteToStorage`	Writes the content of this name space to the specified storage-unit view.

Exists

The `Exists` method returns a Boolean value that indicates whether this name space contains an entry with the specified key.

```
ODBoolean Exists (in ODISOStr key);
```

`key` The key for the requested entry.

return value `kODTrue` if the key exists within this name space, otherwise `kODFalse`.

GetName

The `GetName` method returns the unique name of this name space.

```
ODISOStr GetName ();
```

return value The name of this name space.

DISCUSSION

When you create a name space, you specify its name as a parameter to the name-space manager's `CreateNameSpace` method.

When you no longer need the returned name, you should deallocate it.

SEE ALSO

The `ODISOStr` type (page 845).
The `ODNameSpaceManager::CreateNameSpace` method (page 422).

GetParent

The `GetParent` method returns a reference to the parent name space of this name space.

```
ODNameSpace GetParent ();
```

return value A reference to the parent name space, or `kODNULL` if this name space has no parent.

DISCUSSION

When you create a name space, you specify its parent name space as a parameter to the name-space manager's `CreateNameSpace` method.

SEE ALSO

The `ODNameSpaceManager::CreateNameSpace` method (page 422).

GetType

The `GetType` method returns the type of this name space.

```
ODNSTypeSpec GetType ();
```

return value The type of this name space. The type must be either object name space (`kODNSDataTypeODObject`) or value name space (`kODNSDataTypeODValue`).

DISCUSSION

When you create a name space, you specify its type as a parameter to the name-space manager's `CreateNameSpace` method.

SEE ALSO

The `ODNameSpace::SetType` method (page 418).
The `ODNameSpaceManager::CreateNameSpace` method (page 422).

ReadFromStorage

The `ReadFromStorage` method reads the content of this name space from the specified storage-unit view.

```
void ReadFromStorage (in ODStorageUnitView view);
```

`view` A reference to the storage-unit view from which the data is to be read.

DISCUSSION

You call this method to read in the content of a name space that was written to a storage unit using the `WriteToStorage` method.

The content of this name space is stored in the storage unit as a single stream of data. The focus context for the specified storage-unit view should be the value from which the name space is to be read.

This method does not remove the existing content of this name space, but it will overwrite any entry that has the same key as an entry from the storage unit. After this method executes successfully, this name space contains new entries corresponding to entries read from the storage unit.

EXCEPTIONS

`kODErrInvalidNSName` The stored name does not match the name of this name space.

SEE ALSO

The `ODNameSpace::WriteToStorage` method (page 419).
The `ODStorageUnitView` class (page 700).

SetType

The `SetType` method sets the type of this name space.

```
void SetType (in ODNSTypeSpec type);
```

`type` The type of this name space. The type must be either object name space (`kODNSDataTypeODObject`) or value name space (`kODNSDataTypeODValue`).

DISCUSSION

This method is called by subclasses of the `ODNameSpace` class to set the type of this name space. Your part should not call this method directly.

SEE ALSO

The `ODNameSpace::GetType` method (page 417).

Unregister

The `Unregister` method removes the specified entry from this name space.

```
void Unregister (in ODISOStr key);
```

`key` The key for the entry to be removed.

DISCUSSION

If this name space does not contain an entry with the specified key, this method returns without raising an exception. This method does not search the parent name space for the key.

SEE ALSO

The `ODISOStr` type (page 845).
The `ODObjectNameSpace::Register` method (page 438).
The `ODValueNameSpace::Register` method (page 795).

WriteToStorage

The `WriteToStorage` method writes the content of this name space to the specified storage-unit view.

```
void WriteToStorage (in ODStorageUnitView view);
```

`view`	A reference to the storage-unit view to which the data is to be written.

DISCUSSION

You call this method to write the content of this name space as a single stream of data. The focus context for the specified storage-unit view should be the value to which the name space is to be written.

You can read the content of this name space by calling the `ReadFromStorage` method.

EXCEPTIONS

This method may throw platform-specific exceptions.

SEE ALSO

The `ODNameSpace::ReadFromStorage` method (page 417).
The `ODStorageUnitView` class (page 700).

ODNameSpaceManager

Superclasses `ODObject`
Subclasses none

An object of the `ODNameSpaceManager` class manages the creation and deletion of name spaces among OpenDoc and any associated parts.

Description

When you open a document, the session object creates a single name-space manager object. All parts of that document share the name-space manager; you can obtain a reference to it by calling the session object's `GetNameSpaceManager` method (page 586).

A part creates a name space by calling the name-space manager's `CreateNameSpace` method (page 422). Name spaces may be arranged hierarchically to facilitate searches among multiple name spaces. To obtain a reference to an existing name space, a part calls the name-space manager's `HasNameSpace` method (page 424).

For additional information related to name spaces, see descriptions of the name-space classes `ODNameSpace` (page 414), `ODObjectNameSpace` (page 436), and `ODValueNameSpace` (page 793).

Methods

This section presents summary descriptions of the `ODNameSpaceManager` methods, followed by detailed descriptions in alphabetical order.

`CreateNameSpace`	Creates a new name space.
`DeleteNameSpace`	Deletes the specified name space.

`HasNameSpace`	Checks whether the specified name space exists and, if so, returns a reference to it.

CreateNameSpace

The `CreateNameSpace` method creates a new name space.

```
ODNameSpace CreateNameSpace (
                              in ODISOStr spaceName,
                              in ODNameSpace inheritsFrom,
                              in ODULong numExpectedEntries,
                              in ODNSTypeSpec type);
```

`spaceName` A unique name for the new name space.

`inheritsFrom`
A reference to the parent name space of the new name space, or `kODNULL` if this name space has no parent.

`numExpectedEntries`
The expected number of entries to be stored in the new name space.

`type` The type of the new name space. The type must be either object name space (`kODNSDataTypeODObject`) or value name space (`kODNSDataTypeODValue`).

return value A reference to the newly created name space, or `kODNULL` if a name space already exists with the specified name.

DISCUSSION

You call this method to create either a new object name space or a new value name space, as specified by the `type` parameter. The `spaceName` parameter should specify a unique name for the new name space.

If the `inheritsFrom` parameter is not null, it should specify a parent name space of the same type as the one being created.

Because the current implementation of name spaces uses a hash table, the value you specify in the `numExpectedEntries` parameter should be a prime number to make the hashing algorithm much more efficient. Choose a prime number close to, and greater than, the expected number of entries. The name space can accommodate more entries than you specify (at the expense of performance), but it cannot grow in size.

SEE ALSO

The `ODISOStr` type (page 845).
The `ODNameSpaceManager::DeleteNameSpace` method (page 423).
The `ODNameSpaceManager::HasNameSpace` method (page 424).
The `ODNameSpace` class (page 414).
The `ODObjectNameSpace` class (page 436).
The `ODValueNameSpace` class (page 793).

DeleteNameSpace

The `DeleteNameSpace` method deletes the specified name space.

```
void DeleteNameSpace (in ODNameSpace theNameSpace);
```

`theNameSpace`
A reference to the name space to be deleted.

DISCUSSION

You call this method to delete the specified name space. Deleting a name space removes it from the name-space manager's internal tables and deallocates the memory associated with the object.

If the specified name space was a parent to any other name spaces, those name spaces no longer have a parent name space.

SEE ALSO

The `ODNameSpaceManager::CreateNameSpace` method (page 422).

HasNameSpace

The `HasNameSpace` method checks whether the specified name space exists and, if so, returns a reference to it.

```
ODNameSpace HasNameSpace (in ODISOStr spaceName);
```

`spaceName` The name of the name space to find.

return value A reference to the name space if it exists, otherwise `kODNULL`.

DISCUSSION

You can call this method to check whether a name space with the specified name exists or to obtain a reference to the name space with the specified name.

When you create a name space, you specify its name as a parameter to the name-space manager's `CreateNameSpace` method.

SEE ALSO

The `ODISOStr` type (page 845).
The `ODNameSpaceManager::CreateNameSpace` method (page 422).

ODObject

Superclasses none

Subclasses all OpenDoc classes

The `ODObject` class is the superclass of all OpenDoc classes; it defines features that all subclasses support, such as memory recovery and extensions.

Description

The `ODObject` class defines a general memory recovery system through the `Purge` method (page 428). It also defines an extension mechanism that allows subclasses to extend the functionality of objects. Subclasses are responsible for defining the extensions they support. For more information on extensions, see the `ODExtension` class description (page 208).

You should never instantiate `ODObject`; you can create an object of an `ODObject` subclass by calling the appropriate factory method, if a factory method exists.

Methods

This section presents summary descriptions of the `ODObject` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Object Characteristics

`IsEqualTo`	Returns a Boolean value that indicates whether the specified object is equal to this object.

Extensions

`AcquireExtension`	Returns a reference to the specified extension object.

`HasExtension`	Returns a Boolean value that indicates whether the object supports the specified extension.
`ReleaseExtension`	Releases the specified extension object.

Memory Management

`Purge`	Releases any unneeded memory during low-memory situations.

Debugging

`SubClassResponsibility`
Raises an exception to indicate that a subclass failed to override a required method.

AcquireExtension

The `AcquireExtension` method returns a reference to the specified extension object.

```
ODExtension AcquireExtension (in ODType extensionName);
```

`extensionName`
The name of the extension to acquire. The value of `extensionName` must be one of the following: `kODExtSemanticInterface`, `kODSettingsExtension`, or a part-specific string.

return value A reference to the specified extension object.

DISCUSSION

Your part calls this method to obtain a reference to an extension object with the specified extension type. If the subclass does not support the specified extension, this method raises an exception. The `ODObject` class has no inherent extensions of its own and therefore always raises an exception.

Your override of this method should increment the reference count of the returned extension. When you have finished using that extension, you should call its `Release` method.

OVERRIDING

Your subclass of `ODPart` can override this method if your part supports extensions. If your part does not support the specified extension, your override method must call its inherited `AcquireExtension` method at the end of your implementation.

EXCEPTIONS

`kODErrUnsupportedExtension`
: The specified extension is not supported by this object.

SEE ALSO

The `ODObject::HasExtension` method (page 427).
The `ODObject::ReleaseExtension` method (page 429).
The `ODPart` class (page 445).

HasExtension

The `HasExtension` method returns a Boolean value that indicates whether the object supports the specified extension.

```
ODBoolean HasExtension (in ODType extensionName);
```

`extensionName`
: The name of the extension to look for.

return value
: `kODTrue` if the object supports the specified extension, otherwise `kODFalse`.

DISCUSSION

Your part can call this method before accessing an object's extension.

The `ODObject` class has no inherent extensions of its own and always returns `kODFalse`.

OVERRIDING

Your subclass of `ODPart` can override this method if your part supports extensions. If your part does not support the specified extension, your override method must call its inherited `HasExtension` method at the end of your implementation.

SEE ALSO

The `ODObject::AcquireExtension` method (page 426).
The `ODObject::ReleaseExtension` method (page 429).
The `ODPart` class (page 445).

IsEqualTo

The `IsEqualTo` method returns a Boolean value that indicates whether the specified object is equal to this object.

```
ODBoolean IsEqualTo (in ODObject object);
```

`object`	A reference to an object to compare to this object.
return value	`kODTrue` if the objects are the same, otherwise `kODFalse`.

DISCUSSION

You should call this method whenever you need to compare objects for equality. You should never use the equal or not equal operators (for example, the C++ == and `!=` operators) to compare references of two objects.

Purge

The `Purge` method releases any unneeded memory during low-memory situations.

```
ODSize Purge (in ODSize size);
```

`size`	The number of bytes needed by OpenDoc, expressed as an unsigned 32-bit value.
return value	The number of bytes that were released by this object.

DISCUSSION

Your part may call this method, but in general, OpenDoc calls this method in low-memory situations to free any caches, noncritical buffers, or objects; you should not allocate memory for this operation.

Because the `ODObject` class does not allocate (or deallocate) any memory, it always returns the value 0.

OVERRIDING

Every subclass of `ODObject` can override this method and should do so if it creates caches and temporary buffers. Your subclass of `ODPart` must override this method or risk running out of available memory. Your override method must call its inherited `Purge` method at some point in your implementation (it does not matter where). You should save the size value returned by the inherited `Purge` method, and then add it to the size value returned by your override method to determine the amount of memory actually released.

SEE ALSO

The `ODPart` class (page 445).

ReleaseExtension

The `ReleaseExtension` method releases the specified extension object.

```
void ReleaseExtension (in ODExtension extension);
```

`extension`	A reference to an extension.

DISCUSSION

OpenDoc or an extension object client calls this method to release an extension object that was previously acquired using the `AcquireExtension` method. If the extension was not previously acquired by the caller, the `ReleaseExtension` method raises an exception. After this method executes successfully, the specified extension object is no longer guaranteed to be valid.

The `ODObject` class has no inherent extensions of its own and therefore always raises an exception.

OVERRIDING

Your subclass of `ODPart` can override this method if your part supports extensions. If your part does not support the specified extension, your override method must call its inherited `ReleaseExtension` method at the end of your implementation.

EXCEPTIONS

`kODErrUnsupportedExtension`
: This object does not recognize the extension object and therefore cannot release it.

SEE ALSO

The `ODObject::AcquireExtension` method (page 426).
The `ODObject::HasExtension` method (page 427).
The `ODPart` class (page 445).

SubClassResponsibility

The `SubClassResponsibility` method raises an exception to indicate that a subclass failed to override a required method.

```
void SubClassResponsibility ();
```

DISCUSSION

OpenDoc calls this method to indicate that a subclass that should have overridden a particular method failed to do so.

EXCEPTIONS

`kODErrSubClassResponsibility`
: The called method should have been, but was not, overridden by the subclass of the class that defines the method.

ODObjectIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODObjectIterator` class provides access to the entries of an object name space.

Description

You use an object iterator to apply an operation to all entries of an object name space. For example, you might use an object iterator to compile a complete list of all the part editors that support certain part types.

Your part creates an object iterator object by calling the object name space's `CreateIterator` method (page 437), which returns a reference to an object iterator object. The iterator performs an unordered traversal of the name space.

While you are using an object iterator, you should not modify or delete the name space. You must postpone adding entries to or removing entries from the name space until after you have deleted the iterator.

For information related to object name spaces, see the `ODObjectNameSpace` class description (page 436). For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODObjectIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and obtains the first entry in the name space.
`Next`	Obtains the next entry in the name space.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and obtains the first entry in the name space.

```
void First (out ODISOStr key,
            out ODObject object,
            out ODULong objectLength);
```

`key` A pointer to an ISO string representing the key in the first entry in the name space, or `kODNULL` for an empty name space.

`object` A reference to the object in the first entry in the name space.

`objectLength` The actual size of the specified object, in bytes.

DISCUSSION

Your part must call this method before calling this object iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

It is your responsibility to deallocate the ISO string when it is no longer needed. You must also delete the key when it is no longer needed. You do not need to allocate or deallocate any memory for the `object` parameter.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The name space was modified while the iteration was in progress.

SEE ALSO

The `ODISOStr` type (page 845).

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more entries remain in the name space. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found an entry. This method returns `kODFalse` when you have examined all the entries. If the name space is empty, this method always returns `kODFalse`.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The name space was modified while the iteration was in progress.

Next

The `Next` method obtains the next entry in the name space.

```
void Next (out ODISOStr key,
           out ODObject object,
           out ODULong objectLength);
```

`key` A pointer to an ISO string representing the key in the next entry in the name space, or `kODNULL` if you have reached the end of the name space.

`object` A reference to the object in the next entry in the name space.

`objectLength` The actual size of the specified object, in bytes.

DISCUSSION

If your part calls this method before calling this object iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

It is your responsibility to deallocate the ISO string when it is no longer needed. You must also delete the key when it is no longer needed. You do not need to allocate or deallocate any memory for the `object` parameter.

EXCEPTIONS

`kODErrIteratorOutOfSync` The name space was modified while the iteration was in progress.

SEE ALSO

The `ODISOStr` type (page 845).

ODObjectNameSpace

Superclasses `ODNameSpace` → `ODObject`

Subclasses none

An object of the `ODObjectNameSpace` class is a collection of objects, each of which has a unique key to identify the object within the collection.

Description

An object name space allows a part to identify an object using a unique key, which can be passed easily between parts. A part can create an object name space to store a reference to a part object to facilitate communications with its embedded parts.

Your part can create objects of the `ODObjectNameSpace` class by calling the name-space manager's `CreateNameSpace` method (page 422), passing the constant `kODNSDataTypeODObject` for the type of the name space.

Object name spaces can be arranged hierarchically to allow you to search multiple object name spaces for a single key. Searches move from a child object name space to its parent object name space until the entry is found or until there are no more object name spaces to search.

Methods

This section presents summary descriptions of the `ODObjectNameSpace` methods, followed by detailed descriptions in alphabetical order.

`CreateIterator`	Creates an object iterator for the entries in this object name space.
`GetEntry`	Searches for an entry with the specified key and, if it exists, gets a reference to the object associated with that key.

`Register`	Adds a new entry to this object name space.

CreateIterator

The `CreateIterator` method creates an object iterator for the entries in this object name space.

```
ODObjectIterator CreateIterator ();
```

return value A reference to an object iterator for iterating over the entries in this object name space.

DISCUSSION

You call this method when you need to apply an operation to all the entries of this name space.

While you are using the returned object iterator, you must not modify this object name space; in particular, you must not register or unregister entries and you must not delete this object name space.

You must delete the returned object iterator when it is no longer needed.

SEE ALSO

The `ODObjectIterator` class (page 432).

GetEntry

The `GetEntry` method searches for an entry with the specified key and, if it exists, gets a reference to the object associated with that key.

```
ODBoolean GetEntry (in ODISOStr key,
                    out ODObject object);
```

`key`	The key to search for in this object name space.
`object`	If the entry is found, a reference to the object corresponding to the specified key, otherwise `kODNULL`.
return value	`kODTrue` if the entry was found, otherwise `kODFalse`.

DISCUSSION

If the specified key is found, this method sets the `object` output parameter to a reference to the corresponding object. If the specified key is not found in this name space, this method searches the parent name space. Searches proceed from each object name space to its parent until one of the following happens: the entry is found, there is no parent name space to search, or the parent name space is a value name space instead of an object name space.

If the object corresponding to the specified key is a reference-counted object, this method does not increment the object's reference count. For that reason, if you cache that object, you should call its `Acquire` method to increment its reference count and then call its `Release` method when you are finished using it.

SEE ALSO

The `ODISOStr` type (page 845).
The `ODNameSpace::Exists` method (page 415).
The `ODObjectNameSpace::Register` method (page 438).

Register

The `Register` method adds a new entry to this object name space.

```
void Register (in ODISOStr key,
               in ODObject object);
```

`key`	The key for the new entry.
`object`	A reference to the object to associate with the key.

DISCUSSION

The `Register` method stores a reference to the `object` parameter in a new entry in the table. If the specified key already exists within this name space, this method overwrites the old entry with the new one.

When you register a reference-counted object to an object registry, this method does not increment its reference count.

SEE ALSO

The `ODISOStr` type (page 845).
The `ODNameSpace::Unregister` method (page 419).

ODObjectSpec

Superclasses `ODDesc` → `ODObject`

Subclasses none

An object of the `ODObjectSpec` class is a wrapper for an object specifier structure (type `typeObjectSpecifier`), a descriptor structure on the Mac OS platform that describes the location of one or more Apple event objects.

Description

If your application creates and sends Apple events that require the target application to locate Apple event objects, your application must create object specifiers for those events.

For more information on Apple events and the `typeObjectSpecifier` type, see the "Responding to Apple Events" chapter of *Inside Macintosh: Interapplication Communication*. For general information on scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents a summary description of the `ODObjectSpec` method, followed by a detailed description.

`InitODObjectSpec` Initializes this object specifier.

InitODObjectSpec

The `InitODObjectSpec` method initializes this object specifier.

```
void InitODObjectSpec ();
```

DISCUSSION

There is no factory method for the `ODObjectSpec` class; after creating a new object specifier, OpenDoc or your part must call this method to initialize the new object specifier.

ODOSLToken

Superclasses `ODDesc` → `ODObject`
Subclasses none

An object of the `ODOSLToken` class is a wrapper for an OpenDoc token.

Description

OpenDoc owns the format of the data stored in the `ODOSLToken` class—including information about where the token came from and a reference to a descriptor object.

For more information about descriptor structures, see the `ODDesc` class description (page 102). For general information on tokens and scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODOSLToken` methods, followed by detailed descriptions in alphabetical order.

`InitODOSLToken` Initializes this OpenDoc token.

`DuplicateODOSLToken`
Copies all context information of this OpenDoc token and creates a new OpenDoc token that contains a reference to a descriptor object.

DuplicateODOSLToken

The `DuplicateODOSLToken` method copies all context information of this OpenDoc token and creates a new OpenDoc token that contains a reference to a descriptor object.

```
ODOSLToken DuplicateODOSLToken ();
```

return value A reference to the new OpenDoc token.

DISCUSSION

You must call this method whenever you need to copy an OpenDoc token; it ensures that private data in this OpenDoc token gets copied correctly. You might use this method during object resolution when you are referring to a group of objects and you want to make OpenDoc tokens representing each object.

Although technically there is no factory method for the `ODOSLToken` class, the `DuplicateODOSLToken` method could be considered a factory method since it is really the only valid way for you to create a new OpenDoc token.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to allocate the OpenDoc token.

InitODOSLToken

The `InitODOSLToken` method initializes this OpenDoc token.

```
void InitODOSLToken ();
```

DISCUSSION

There is no factory method for the `ODOSLToken` class; after creating a new OpenDoc token, OpenDoc or your part must call this method to initialize the new OpenDoc token.

ODPart

Superclasses `ODPersistentObject` → `ODRefCntObject` → `ODObject`
Subclasses none

An object of a subclass of `ODPart` represents a part editor.

Description

Parts are the building blocks of OpenDoc documents. A compound document can contain one or more parts, each manipulated by a part editor. A part editor is much like an application; it can display and change the content of a part, handle events, perform disk I/O, and accept scripting commands. A part editor directly manipulates the content, the **intrinsic content**, of its part.

The `ODPart` class is an abstract superclass that you must subclass to create your part editor. Your implementation is the executable code that provides the structure and behavior of your parts. When you subclass `ODPart`, you need to override only those methods that represent specific capabilities supported by your part editor.

OpenDoc or another part creates your part object by calling the draft's `CreatePart` method (page 166). OpenDoc or another part can also access a previously created, stored part by calling the draft's `AcquirePart` method (page 154). These methods return a reference to a part object.

Layout and Imaging

OpenDoc provides an environment for managing the geometric relationships among frames and objects when your part draws itself. OpenDoc notifies each part when the part editor must draw the part. Each part is responsible for drawing its own content (including, at certain times, the borders of its embedded frames). A part does not draw the interiors of its embedded frames because they contain the content of other parts (which must draw themselves). Drawing may occur asynchronously, in response to update events, or when

activation or deactivation affects highlighting. A part editor typically draws in the context of a particular facet. The part editor gets the clipping, transform, and layout information from the facet and its frame, and then must make platform-specific graphics calls to actually draw the lines, polygons, text, and so on that make up its part.

A container part can define a set of **container properties** (such as text font, background color, or pen width) that it prefers its embedded parts to adopt to enhance the visual continuity between them. Embedded parts that can read a given container part's container properties can then adopt those properties that are appropriate.

View Type and Presentation

Each frame has a **view type**, the basic visual representation of a part in that frame. Parts must support the standard set of view types (large icon, small icon, thumbnail, and frame view). The **presentation** of a frame describes a particular style of display for a part's content within the frame when its view type is framed. Presentations are part-defined; your part editor determines what types of presentations your part can handle. Your part is notified if its containing part changes the view type or presentation of one of its display frames.

Frame Negotiation

Each part in an OpenDoc document controls the position, size, and shape of its embedded frames. If an embedded part needs to change the size, shape, or number of frames it is displayed in, it must negotiate for that change with its containing part. Frame negotiation allows an embedded part to communicate its needs to its containing part.

Initialization and Storage

Your part editor must be able to initialize its parts, as well as read and write their content to and from persistent storage in its document. The data of each part in a document is kept in at least one **storage unit**, the basic unit of persistent storage, distinct from data in other parts. You use the **contents property** of your storage unit for storing all your part's data.

Your part-editor code should never pass out a pointer to `somSelf`, except to supported extensions and your embedded-frames iterator. Instead, your part must store and pass out a reference to a part wrapper—an object that forwards all method calls to your part. Part wrappers insulate OpenDoc from requiring a direct reference to your part. A reference to a stored part wrapper is specified in your part's `InitPart` method (page 509) or `InitPartFromStorage` method (page 510). The reference is defined for the lifetime of your part; once set, it cannot be changed.

Event Handling

OpenDoc handles events such as mouse clicks and keystrokes, menu commands, activation and deactivation of windows, and other platform-specific events. Part editors do not receive events directly from the operating system; rather, OpenDoc receives events, interprets them, and dispatches them, using the dispatcher, to the appropriate part editor's `HandleEvent` method (page 506). Based on information in an event, your part editor might update your part, open or close windows, perform editing operations, transfer data, perform menu commands, or perform other operations. For more information on event handling in OpenDoc, see the chapter on user events in the *OpenDoc Programmer's Guide for the Mac OS*.

Keeping Track of Display Frames and Embedded Frames

Your part can display its content in one or more frames at the same time, displaying either duplicate views or different aspects of the same part. This makes it easy for your part to be visible in several windows or to have several different presentations. Identical views of your part in two or more separate display frames must be **synchronized**, meaning that any editing or other changes to one display frame forces updating of the others. If your part is a containing part, your part calls its embedded frame's `AttachSourceFrame` method (page 466) to initiate synchronization of it embedded frames.

Your part must maintain a list of all frames in which it is displayed. In addition, if your part is a containing part, it must maintain a list of all frames directly embedded within it. For embedded frames, you may use whatever format for that list that works best for your implementation, but you must provide an embedded-frames iterator (a subclass of `ODEmbeddedFramesIterator`) to allow access to the frames.

For efficient memory usage, your part does not need to keep frame objects in memory for all its embedded frames or display frames. Instead, it can release frames as they become invisible through scrolling and then get them back from the draft as they become visible. For more information related to frame objects, see the `ODFrame` class description (page 288).

Part Kind and Binding

Parts store their data in formats defined by **part kind**, a typing scheme analogous to file type. OpenDoc uses the concept of part kind to determine what part editor is to be associated with a given part in a document. You can design your part editor to manipulate more than one part kind, and your part can be stored with multiple representations of its data, each of a different part kind. To maximize fidelity in case your part is later edited by a different part editor, your part editor must always store its part kinds in order from highest fidelity to lowest.

Linking

Your part uses linking to export updatable data to another part, or to incorporate or embed an updatable copy of another part's data into your part. If your part is the source of a link, you interact with a link-source object; if your part is the destination of a link, you interact with a link object.

Overriding ODPart Methods

The following table lists the methods of `ODPart` that you must override to have a functioning part editor, as well as those that you can optionally override to participate in specific protocols. Note that some protocols, such as layout, imaging, and activation, are required of all part editors, and you must override some or all of the methods associated with them. Other protocols, such as embedding or undo, are not required, and you need not override any of their

methods if your parts do not participate. It is, of course, strongly recommended that your parts participate in all protocols.

Required and optional ODPart overrides

Protocol	Required overrides	Optional overrides

Layout	`AttachSourceFrame`
	`ContainingPartPropertiesUpdated`
	`DisplayFrameAdded`
	`DisplayFrameClosed`
	`DisplayFrameConnected`
	`DisplayFrameRemoved`
	`FacetAdded`
	`FacetRemoved`
	`FrameShapeChanged`
	`GeometryChanged`
	`Open`
	`SequenceChanged`
		`AcquireContainingPartProperties`*
		`RevealFrame`*

Imaging	`CanvasChanged`
	`Draw`
	`GetPrintResolution`
	`HighlightChanged`
	`PresentationChanged`
	`ViewTypeChanged`
		`AdjustBorderShape`*
		`CanvasUpdated`*

Activation	`AbortRelinquishFocus`
	`BeginRelinquishFocus`
	`CommitRelinquishFocus`
	`FocusAcquired`
	`FocusLost`	

User events	`AdjustMenus` `HandleEvent`	

Required and optional ODPart overrides (continued)

Protocol	Required overrides	Optional overrides
Storage	`CloneInto`†	`somInit`†
	`ClonePartInfo`	`somUninit`†
	`Externalize`†	
	`ExternalizeKinds`	
	`InitPart`	
	`InitPartFromStorage`	
	`ReadPartInfo`	
	`WritePartInfo`	
Binding	`ChangeKind`	
Memory management	`ReleaseAll`†	`Acquire`†
		`Purge`†
		`Release`†
Linking	`LinkStatusChanged`	`CreateLink`
		`EditInLinkAttempted`*
		`FulfillPromise`
		`LinkUpdated`
		`RevealLink`
Embedding		`CreateEmbeddedFramesIterator`*
		`EmbeddedFrameUpdated`*
		`RemoveEmbeddedFrame`*
		`RequestEmbeddedFrame`*
		`RequestFrameShape`*
		`UsedShapeChanged`*
Clipboard		`FulfillPromise`
Drag and drop		`DragEnter`
		`DragLeave`
		`DragWithin`
		`Drop`
		`DropCompleted`
		`FulfillPromise`

Required and optional ODPart overrides (continued)

Protocol	Required overrides	Optional overrides

Undo		`DisposeActionState`
		`ReadActionState`
		`RedoAction`
		`UndoAction`
		`WriteActionState`

Extensions		`AcquireExtension`† `HasExtension`† `ReleaseExtension`†
Semantic events		`EmbeddedFrameSpec`*

* Required of all parts that support embedding
† Defined in a superclass of `ODPart`; see "Overriding Inherited Methods"

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODPart`. You need to override only those methods that represent specific capabilities supported by your part editor.

somInit

The `somInit` method initializes the instance variables in a SOM object; it is inherited from the `SOMObject` class.

When you subclass `ODPart`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in your part. The SOM library calls this method when your part is created. You must not do anything that might fail in this method. This limits you to operations like setting pointer variables to null, setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in your part's `InitPart` or `InitPartFromStorage` method.

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

When you subclass `ODPart`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for your part, including any storage related to additional instance variables initialized in your part. The SOM library calls this method when your part is deleted; this method must not fail.

HasExtension

The `HasExtension` method returns a Boolean value that indicates whether an object supports a specified extension; it is inherited from the `ODObject` class.

```
ODBoolean HasExtension (in ODType extensionName);
```

When you subclass `ODPart`, you can override this method if your part supports extensions. If your part does not support the specified extension, your override method must call its inherited method at the end of your implementation. The client of the extension object must be prepared in the event that the specified extension is not supported.

Your override of this method should return `kODTrue` if your part supports the specified extension, otherwise `kODFalse`. A client of an extension object calls this method to determine whether your part supports extensions, for example, scripting.

AcquireExtension

The `AcquireExtension` method returns a reference to a specified extension object; it is inherited from the `ODObject` class.

```
ODExtension AcquireExtension (in ODType extensionName);
```

When you subclass `ODPart`, you can override this method if your part supports extensions. If your part does not support the specified extension, your

override method must call its inherited method at the end of your implementation. The client of the extension object must be prepared in the event that the specified extension is not supported.

Your override of this method should return a reference to your part's extension object (for example, a semantic interface object). A client of an extension object calls this method to obtain a reference to the specified extension.

ReleaseExtension

The `ReleaseExtension` method releases a specified extension object; it is inherited from the `ODObject` class.

```
void ReleaseExtension (in ODExtension extension);
```

When you subclass `ODPart`, you can override this method if your part supports extensions. If your part does not support the specified extension, your override method must call its inherited method at the end of your implementation. The client of the extension object must be prepared in the event that the specified extension is not supported.

Your override of this method should release the specified extension object (acquired when your part's `AcquireExtension` method was called). A client of an extension object calls this method when it finishes working with your part's extension.

Acquire

The `Acquire` method increments an object's reference count by 1; it is inherited from the `ODRefCntObject` class.

```
void Acquire ();
```

When you subclass `ODPart`, you can override this method if your part performs any specific actions when its reference count is incremented. Your override method must call its inherited method at the beginning of your implementation.

The inherited `Acquire` method increments your part's reference count by 1. A part editor calls this method when it stores a reference to your part. When the

reference is replaced or removed, a part editor calls your part's `Release` method.

Release

The `Release` method decrements an object's reference count by 1 and tells the draft to release the object from memory if the object's reference count becomes 0; it is inherited from the `ODRefCntObject` class.

```
void Release ();
```

When you subclass `ODPart`, you can override this method if your part needs to release an object and reclaim valuable resources like memory. Your override method must call its inherited method at the beginning of your implementation.

The inherited `Release` method decrements your part's reference count by 1. The inherited method may delete your part from memory (if your part's reference count becomes 0). A part editor calls this method when it no longer needs a reference to your part.

ReleaseAll

The `ReleaseAll` method releases all transitory references to other reference-counted objects; it is inherited from the `ODPersistentObject` class.

```
void ReleaseAll ();
```

When you subclass `ODPart`, you can override this method if your part maintains references to other persistent objects. Your override method must call its inherited method at the end of your implementation.

Your override of this method should release all persistent objects it points to, remove any link specifications your part has written to the clipboard, relinquish all foci, and clear any undo actions it has written to the undo stack. For any extension that cannot be deleted because of a positive reference count, your override method must also call its base extension's `BaseRemoved` method. If your part's embedded frames are being traversed by an embedded-frames iterator when your part is deleted, your override method must also call your embedded-frames iterator's `PartRemoved` method.

OpenDoc calls this method once before your part is deleted, for example, when your part's document is closed. If the part being deleted has a promise, the promise is resolved before the part is deleted.

CloneInto

The `CloneInto` method clones a persistent object by copying its data into a specified storage unit; it is inherited from the `ODPersistentObject` class.

```
void CloneInto (in ODDraftKey key,
                in ODStorageUnit toSU,
                in ODFrame scope);
```

When you subclass `ODPart`, you must override this method to support data interchange of internal data. Your override method must call its inherited method at the beginning of your implementation. The inherited method clones the `kODPropCreateDate`, `kODPropModDate`, and `kODPropModUser` properties.

Your override of this method should write your part's data into the specified storage unit and clone any additional objects to which your part has strong and weak persistent references and that are within the specified scope. OpenDoc calls this method if your part is being copied to the clipboard, the drag-and-drop object, or a link-source object.

Externalize

The `Externalize` method writes a persistent object to storage; it is inherited from the `ODPersistentObject` class.

```
void Externalize ();
```

When you subclass `ODPart`, you can override this method if your part maintains persistent content. Your override method must call its inherited method at the beginning of your implementation.

Your override of this method should write your part's persistent state to the properties and values your part created when it initialized itself in your part's `InitPart` method (page 509); it should write sufficient information to allow your `InitPartFromStorage` method (page 510) to restore your part to its

current state. Your part editor calls this method when your part stores its content data.

Purge

The `Purge` method frees memory on request; it is inherited from the `ODObject` class.

```
ODSize Purge (in ODSize size);
```

Every subclass of `ODObject` can override this method and should do so if it creates caches and temporary buffers. When you subclass `ODPart`, you must override this method or risk running out of available memory. Your override method must call its inherited method at some point in your implementation (it does not matter where). You should save the size value returned by the inherited method because you will need it to compute the value to return from your override method.

Your override of this method should free any caches, noncritical buffers, or objects (up to the amount of memory specified). If absolutely necessary, your part can first write any data it needs to persistent storage, although that is not recommended because writing to storage requires additional memory allocation. Your override of this method should add the number of bytes actually freed to the number returned by the inherited method and return the sum as the total amount of memory released. OpenDoc calls this method in low-memory situations; you should not allocate memory for this operation.

Methods

This section presents summary descriptions of the `ODPart` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Initialization and Storage

`InitPart`	Initializes this part object.
`InitPartFromStorage`	Initializes this part object from its stored data.
`ReadPartInfo`	Should read the part info data for a display frame of this part from the specified storage-unit view object.

`WritePartInfo`	Should write the part info data for a display frame of this part into the specified storage-unit view object.
`ClonePartInfo`	Should clone a display frame's part info data into the specified storage-unit view object.
`ExternalizeKinds`	Should write to storage a representation for each part kind within the specified kind list that this part supports.

Binding

`ChangeKind`	Should change this part's preferred kind.
`IsRealPart`	Returns a Boolean value that indicates whether this part is an actual part (as opposed to a part wrapper).
`GetRealPart`	Returns a reference to a part object encapsulated by the part wrapper.
`ReleaseRealPart`	Releases the part object encapsulated by the part wrapper.

Imaging

`CanvasChanged`	Should transfer asynchronous imaging to a new canvas.
`CanvasUpdated`	Should copy the content of the specified canvas (owned by this part) to its parent canvas.
`Draw`	Should draw this part within the area that needs updating in the specified facet.
`GetPrintResolution`	Should return the minimum desired resolution, in dots per inch, required for printing the content of the specified display frame.
`HighlightChanged`	Should update the highlight state of the specified facet of this part.

Facet Manipulation

`FacetAdded`	Should notify this part that a facet has been added to one of its display frames.
`FacetRemoved`	Should notify this part that a facet has been removed from one of its display frames.

`GeometryChanged` Should notify this part that the clip shape or external transform (or both) of one of this part's facets has changed.

Frame Negotiation

`FrameShapeChanged` Should notify this part that the frame shape of one of its display frames has changed.

`RequestEmbeddedFrame` Should create a new display frame for the specified embedded part.

`RequestFrameShape` Should negotiate a new frame shape for the specified frame embedded in this part.

Display Frame Manipulation

`AttachSourceFrame` Should associate a source frame with a display frame for this part.

`DisplayFrameAdded` Should add the specified frame to this part's internal list of display frames.

`DisplayFrameClosed` Should notify this part that one of its display frames has been closed.

`DisplayFrameConnected` Should add the specified frame to this part's internal list of display frames.

`DisplayFrameRemoved` Should remove the specified frame from this part's internal list of display frames.

`PresentationChanged` Should notify this part that the presentation of one of its display frames has changed.

`ViewTypeChanged` Should notify this part that the view type of one of its display frames has changed.

`SequenceChanged` Should notify this part that the sequencing of this part's display frame within its frame group has changed.

`Open` Should create or activate a window in which a frame of this part is the root frame.

Embedded Frame Manipulation

`AdjustBorderShape` Should adjust the shape of an embedded frame's active frame border.

`CreateEmbeddedFramesIterator` Should create an embedded-frames iterator to give callers access to all embedded frames of this part.

`RemoveEmbeddedFrame` Should remove the specified embedded frame from this part's content.

`EmbeddedFrameSpec` Should create an object specifier for the specified embedded frame in this part.

`RevealFrame` Should make the specified embedded frame visible by scrolling it into view.

`UsedShapeChanged` Should notify this part that the used shape of one of its embedded frames has changed.

Focus Manipulation

`AbortRelinquishFocus` Should reestablish this part's ownership of the focus specified in a previous call to this part's `BeginRelinquishFocus` method.

`BeginRelinquishFocus` Should return a Boolean value that indicates whether this part is willing to relinquish the requested focus and should prepare this part to do so.

`CommitRelinquishFocus` Should complete this part's relinquishing of ownership of the specified focus.

`FocusAcquired` Should notify this part that one of its display frames has acquired the specified focus.

`FocusLost` Should notify this part that one of its display frames has lost the specified focus.

Event Handling

`AdjustMenus` Should prepare this part's menus for display.

`HandleEvent` Should attempt to handle the specified user event.

Undo

`RedoAction`	Should redo the specified action.
`UndoAction`	Should undo the specified action.
`DisposeActionState`	Should dispose of the undo action data.
`ReadActionState`	Should read the undo action data from the specified storage-unit view object.
`WriteActionState`	Should write the undo action data into the specified storage-unit view object.

Linking

`CreateLink`	Should create a link-source object for this part.
`EmbeddedFrameUpdated`	Should update any of this part's link-source objects affected by a change to the specified embedded frame.
`EditInLinkAttempted`	Should return a Boolean value that indicates whether this part maintains the destination of a link that includes the embedded frame.
`LinkStatusChanged`	Should notify this part that the link status of one of its display frames has changed.
`LinkUpdated`	Should replace the content at each destination of a link with new content from an updated link object.
`RevealLink`	Should show the content at the source of a link.

Data Interchange

`DragEnter`	Should begin tracking a drag operation.
`DragWithin`	Should track a drag operation and provide graphical feedback regarding possible drop targets.
`DragLeave`	Should finish tracking a drag operation and deactivate this part from drag tracking.
`Drop`	Should move or copy the dragged data into this part.
`DropCompleted`	Should notify this part that a drop operation, resulting from an asynchronous drag initiated from this part, is complete.

`FulfillPromise` Should fulfill a promise by providing the content data the promise represents.

Container Properties

`AcquireContainingPartProperties`
Should write into a storage unit the container properties this part associates with the specified embedded frame and then return the storage unit.

`ContainingPartPropertiesUpdated`
Should notify this part that its containing part's container properties have changed.

AbortRelinquishFocus

The `AbortRelinquishFocus` method should reestablish this part's ownership of the focus specified in a previous call to this part's `BeginRelinquishFocus` method.

```
void AbortRelinquishFocus (in ODTypeToken focus,
                           in ODFrame ownerFrame,
                           in ODFrame proposedFrame);
```

`focus` A tokenized string representing the focus type that was to be relinquished, expressed as a 32-bit value.

`ownerFrame`
A reference to a display frame that currently possesses the focus.

`proposedFrame`
A reference to a frame that originally requested the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

A request for a focus set is conditional; if any focus owner is unwilling to relinquish a focus, then none are acquired. Your part's `AbortRelinquishFocus` method should give those focus owners who have indicated willingness to relinquish the focus an opportunity to back out of changes initiated when OpenDoc first called your part's `BeginRelinquishFocus` method. When a request for a focus set fails because one of the focus owners will not relinquish ownership, OpenDoc aborts the request to relinquish all of the foci by calling each part's `AbortRelinquishFocus` method.

Your part's `AbortRelinquishFocus` method should return your part to the state where it possessed the focus before continuing to use that focus; how it does this is dependent on your part editor and the type of focus relinquished. Calling this method should have no effect if your part's `BeginRelinquishFocus` method returned `kODFalse`.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODPart::BeginRelinquishFocus` method (page 467).
The `ODPart::CommitRelinquishFocus` method (page 472).
The `ODSession::Tokenize` method (page 598).

AcquireContainingPartProperties

The `AcquireContainingPartProperties` method should write into a storage unit the container properties this part associates with the specified embedded frame and then return the storage unit.

```
ODStorageUnit AcquireContainingPartProperties (
                                        in ODFrame frame);
```

`frame`	A reference to an embedded frame of this part.
return value	A reference to the storage unit that contains the container properties for the specified frame.

DISCUSSION

The part embedded in this part calls this method to request the container properties that this part associates with the embedded part's display frame.

Before returning the storage unit object, your part's `AcquireContainingPartProperties` method should call the storage unit object's `Acquire` method. When the caller has finished using the returned storage unit object, it should call the storage unit object's `Release` method.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.
`kODErrInvalidFrame`	The specified frame is not an embedded frame of this part.

SEE ALSO

The `ODPart::ContainingPartPropertiesUpdated` method (page 474).

AdjustBorderShape

The `AdjustBorderShape` method should adjust the shape of an embedded frame's active frame border.

```
ODShape AdjustBorderShape (in ODFacet embeddedFacet,
                           in ODShape shape);
```

`embeddedFacet`
: A reference to an embedded facet for which the active frame border must be adjusted.

`shape`
: A reference to a shape object defining the current active frame border, or `kODNULL` if the active frame border is transferred to an embedded part of another part.

return value
: A reference to a revised shape object to use for the embedded frame's active frame border, clipped by this part's content and used shape.

DISCUSSION

OpenDoc calls this method when a frame embedded in this part has the selection focus. OpenDoc draws the active frame border, associated with the frame's facet, on the border of the facet's frame. Your part's `AdjustBorderShape` method may be called multiple times for a containing part that has several facets for the same embedded frame.

Your part's `AdjustBorderShape` method should clip the provided shape to account for content elements (including other embedded frames) that obscure portions of the active frame border and then return the shape. Before returning the shape object, your part's `AdjustBorderShape` method should call the shape object's `Acquire` method (unless it modifies the requested shape object and returns that same object). When the caller has finished using the returned shape object, it should call the shape object's `Release` method.

Your part's `AdjustBorderShape` method should also adjust the clip shape of your part's facet to account for portions of the active frame border that obscure your content.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.
`kODErrInvalidFacet`	The specified facet is not an embedded facet of this part.

AdjustMenus

The `AdjustMenus` method should prepare this part's menus for display.

```
void AdjustMenus (in ODFrame frame);
```

`frame`	A reference to a display frame whose menus should be displayed.

DISCUSSION

OpenDoc calls this method when this part has the menu focus or is the root part and there is a mouse-down event in the menu bar.

Your part's `AdjustMenus` method should perform any actions necessary to enable and disable menu items as appropriate to its current state. Your part's `AdjustMenus` method can use the `ODMenuBar` method to change the appearance of your menu items, or it can make platform-specific calls to directly enable, disable, mark, or change the text of your menu items.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

AttachSourceFrame

The `AttachSourceFrame` method should associate a source frame with a display frame for this part.

```
void AttachSourceFrame (in ODFrame frame,
                        in ODFrame sourceFrame);
```

`frame` A reference to a display frame for this part.

`sourceFrame`
A reference to a display frame to be used as the source for the display frame specified in the `frame` parameter.

DISCUSSION

This part's containing part calls this method. This method tells this part to keep two or more of its display frames synchronized. The containing part calls this method immediately after creating a new display frame for this part.

Your part's `AttachSourceFrame` method should perform any actions necessary to synchronize the frames. If the two frames have identical presentations, your method should copy all embedded frames in the source frame into the other frame (and attach the copied embedded frames to their source frames by calling the embedded part's `AttachSourceFrame` method).

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

BeginRelinquishFocus

The `BeginRelinquishFocus` method should return a Boolean value that indicates whether this part is willing to relinquish the requested focus and should prepare this part to do so.

```
ODBoolean BeginRelinquishFocus (in ODTypeToken focus,
                                in ODFrame ownerFrame,
                                in ODFrame proposedFrame);
```

`focus`	A tokenized string representing the focus type to be relinquished, expressed as a 32-bit value.
`ownerFrame`	A reference to a display frame that currently possesses the focus.
`proposedFrame`	A reference to a frame requesting ownership of the focus.
return value	`kODTrue` if this part is willing to relinquish the requested focus, otherwise `kODFalse`.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method when ownership of the specified focus is requested and this part is the current owner. Upon receiving this call, this part decides whether it can relinquish the specified focus. Each part has part-specific requirements that must be met before it can relinquish a focus. If these requirements are met, this part should be ready to relinquish the focus.

Your part's `BeginRelinquishFocus` method should return either `kODTrue` or `kODFalse`, based on the type of focus and the identities of the frames (current and proposed focus owners) passed in. In most cases, your method should return `kODTrue`. However, your method might return `kODFalse`, for example, if your part is displaying a modal dialog box and another part is requesting the modal focus. In that case, because your part does not want to yield the modal focus until its modal dialog box closes, your method should return `kODFalse`. In situations when your method does return `kODFalse`, then your part's `AbortRelinquishFocus` method does not need to take any action.

Your part's `BeginRelinquishFocus` method should also ensure that, during the process of relinquishing the specified focus and before your part's `CommitRelinquishFocus` method is called, your part does not enter a state where the focus cannot be relinquished.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified owner frame is not a display frame of this part.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODPart::AbortRelinquishFocus` method (page 461).
The `ODPart::CommitRelinquishFocus` method (page 472).
The `ODSession::Tokenize` method (page 598).

CanvasChanged

The `CanvasChanged` method should transfer asynchronous imaging to a new canvas.

```
void CanvasChanged (in ODFacet facet);
```

`facet` A reference to a facet with the new canvas.

DISCUSSION

OpenDoc calls this method if an offscreen canvas is assigned to, changed, or removed from your part's facet while your part is running. Your part must prepare to draw on the new canvas, update the cached information, and alter asynchronous display information.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

CanvasUpdated

The `CanvasUpdated` method should copy the content of the specified canvas (owned by this part) to its parent canvas.

```
void CanvasUpdated (in ODCanvas canvas);
```

`canvas` A reference to a canvas that is updated.

DISCUSSION

Your part's `CanvasUpdated` method is called when an area of a canvas owned by this part changes and needs to be updated. The owning part's facet calls this

method to allow the owning part to copy the image of the offscreen canvas to its parent canvas during updates.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.

ChangeKind

The `ChangeKind` method should change this part's preferred kind.

```
void ChangeKind (in ODType kind);
```

`kind`	The preferred kind to assign to this part.

DISCUSSION

OpenDoc calls this method if the user has chosen a new preferred kind for this part in the Part Info dialog box (for example, a request that a Styled Text part change into an ASCII Text part).

After your part's `ChangeKind` method executes successfully, your part should begin using the specified part kind as its preferred kind and begin manipulating your part's data in the new format. Your part should write the preferred kind into a `kODPropPreferredKind` property in its storage unit and store its part kinds in order from highest fidelity to lowest.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. Your override method should always be implemented, not just implemented in the case of a part supporting multiple kinds.

ClonePartInfo

The `ClonePartInfo` method should clone a display frame's part info data into the specified storage-unit view object.

```
void ClonePartInfo (in ODDraftKey key,
                    in ODInfoType partInfo,
                    in ODStorageUnitView storageUnitView,
                    in ODFrame scope);
```

`key` The draft key identifying a particular cloning operation, expressed as a 32-bit value. The key provides thread-safe access to cloning.

`partInfo` The part info data to clone.

`storageUnitView`
A reference to a storage-unit view object that is focused to the frame's part info property, but not to any value within that property.

`scope` A reference to a display frame that defines the scope of the cloning operation. All storage units cloned must be within the scope of the specified frame.

DISCUSSION

When a document is saved as a copy (using the Save a Copy command from the Document menu) or data is transferred (using drag and drop or linking), OpenDoc may ask your part to save a display frame's part info data to a

particular storage unit. OpenDoc calls this method when it writes the part info data for a display frame of your part.

Your part's `ClonePartInfo` method must get the storage unit associated with the specified storage-unit view and focus it to the value(s) in the frame's part info property as necessary to write the formats it needs.

Your part's `ClonePartInfo` method should write out the part info data. Your part's `ClonePartInfo` method should also clone any additional objects to which your part has strong persistent references and that are within the specified display frame's scope.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODDraftKey` type (page 872).
The `ODInfoType` type (page 853).
The `ODPart::ReadPartInfo` method (page 517).
The `ODPart::WritePartInfo` method (page 532).

CommitRelinquishFocus

The `CommitRelinquishFocus` method should complete this part's relinquishing of ownership of the specified focus.

```
void CommitRelinquishFocus (in ODTypeToken focus,
                            in ODFrame ownerFrame,
                            in ODFrame proposedFrame);
```

`focus` A tokenized string representing the focus type to be relinquished, expressed as a 32-bit value.

`ownerFrame`
A reference to a display frame that currently possesses the focus.

`proposedFrame`
A reference to a frame that requested the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

Your part's `CommitRelinquishFocus` method should complete the process initiated when OpenDoc first called your part's `BeginRelinquishFocus` method. If all focus owners in a given request for a focus set are willing to relinquish focus, OpenDoc calls each part's `CommitRelinquishFocus` method.

Your part's `CommitRelinquishFocus` method should perform any actions necessary to relinquish the specified focus to OpenDoc. Some actions depend on the nature and implementation of the part itself, but others are standard. Your part's `CommitRelinquishFocus` method should remove menus or palettes of your part's frame, remove highlighting, and relinquish any external resources associated with the specified focus. Remember that the focus might be moving from one frame to another of the same part, so the exact actions can vary.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODPart::AbortRelinquishFocus` method (page 461).
The `ODPart::BeginRelinquishFocus` method (page 467).
The `ODSession::Tokenize` method (page 598).

ContainingPartPropertiesUpdated

The `ContainingPartPropertiesUpdated` method is called to notify this part that its containing part's container properties have changed.

```
void ContainingPartPropertiesUpdated (
                            in ODFrame frame,
                            in ODStorageUnit propertyUnit);
```

`frame` A reference to a display frame to which the container properties apply.

`propertyUnit`
A reference to a storage unit that contains the changed container properties.

DISCUSSION

This part's containing part calls this method when the container properties it provides for adoption have changed. Your part's `ContainingPartPropertiesUpdated` method should inspect the specified storage unit for container properties that this part can understand. Where applicable, your method should adopt those container properties for its own appearance or behavior. It ignores inapplicable container properties, without generating an exception.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODPart::AcquireContainingPartProperties` method (page 463).

CreateEmbeddedFramesIterator

The `CreateEmbeddedFramesIterator` method should create an embedded-frames iterator to give callers access to all embedded frames of this part.

```
ODEmbeddedFramesIterator CreateEmbeddedFramesIterator (
                                            in ODFrame frame);
```

`frame`	A reference to a display frame whose embedded frames the iterator traverses.
return value	A reference to a new embedded-frames iterator object.

DISCUSSION

Your part's `CreateEmbeddedFramesIterator` method should create, initialize, and return an embedded-frames iterator. Your part calls this method if it needs to apply an operation to all frames directly embedded within a display frame of another part. It is your responsibility to delete the iterator when it is no longer needed.

While you are using an embedded-frames iterator, you should not modify the list of embedded frames for the part. You must postpone adding items to or removing items from the list of embedded frames for the part until after you have deleted the iterator.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.
`kODErrInvalidFrame`	The specified frame is not a display frame of this part.
`kODErrOutOfMemory`	There is not enough memory to allocate the embedded-frames iterator object.

SEE ALSO

The `ODEmbeddedFramesIterator` class (page 198).

CreateLink

The `CreateLink` method should create a link-source object for this part.

```
ODLinkSource CreateLink (in ODByteArray data);
```

`data`	A byte array whose buffer contains the data of a link specification, created earlier by this part, that defines the content of the link source to be linked to.
return value	A reference to a new link-source object to be used to represent the data.

DISCUSSION

OpenDoc calls this method when the user decides to create a link when pasting in or dropping data that originated in this part. If a link already exists to the specified source content, this method should return the existing link-source

object; otherwise, this method should create a new link-source object and copy the source content into it.

The data in a link specification is private to the part writing it. The data is only returned to this part when the `CreateLink` method actually creates a link-source object. If your part creates a link-source object, your part must maintain information about what portion of its content has been linked so that it may update the link-source object when that content data changes.

Before returning the link-source object, your part's `CreateLink` method should call the link-source object's `Acquire` method. When the caller has finished using the returned link-source object, it should call the link-source object's `Release` method.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotLink`	This part does not support linking.
`kODErrOutOfMemory`	There is not enough memory to allocate the link-source object.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODLinkSource` class (page 361).
The `ODLinkSpec` class (page 379).

DisplayFrameAdded

The `DisplayFrameAdded` method should add the specified frame to this part's internal list of display frames.

```
void DisplayFrameAdded (in ODFrame frame);
```

`frame` A reference to a display frame to be added.

DISCUSSION

OpenDoc calls this method only when a new display frame has been added to this part during frame object initialization.

Your part's `DisplayFrameAdded` method should perform any actions necessary to handle the addition of the specified frame to your part's internal list of display frames. Some actions depend on the nature and implementation of the part itself, but others are standard, such as adding the new frame to your part's internal list of display frames, verifying that you can support the frame's view type and presentation, and adding any needed part info data to the frame.

Your part should not perform any layout or imaging tasks as a result of a display frame being added; it should wait until your part's `FacetAdded` method is called, at which time the frame becomes visible.

Your part's `DisplayFrameAdded` method is typically called after OpenDoc or this part's containing part calls a draft's `CreateFrame` method. By contrast, a part's `DisplayFrameConnected` method is called after OpenDoc or this part's containing part calls a draft's `GetFrame` method.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODPart::DisplayFrameConnected` method (page 480).

DisplayFrameClosed

The `DisplayFrameClosed` method is called to notify this part that one of its display frames has been closed.

```
void DisplayFrameClosed (in ODFrame frame);
```

`frame` A reference to a display frame to be closed.

DISCUSSION

OpenDoc calls this method when this part's document or window closes.

Your part's `DisplayFrameClosed` method should update your part's internal list of display frames and other related structures to reflect the removal of the specified display frame from memory, and it should close any embedded frames. In addition, your method should call the arbitrator's `RelinquishFocusSet` method to relinquish any foci owned by the display frame being closed.

Your part's `DisplayFrameClosed` method should not alter a part's stored list of display frames; it reflects only the deletion of the currently instantiated frame object. By contrast, OpenDoc calls a part's `DisplayFrameRemoved` method to permanently remove one of the part's display frames.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODPart::DisplayFrameRemoved` method (page 481).

DisplayFrameConnected

The `DisplayFrameConnected` method should add the specified frame to this part's internal list of display frames.

```
void DisplayFrameConnected (in ODFrame frame);
```

`frame` A reference to a display frame to be connected.

DISCUSSION

OpenDoc calls this method when a previously stored display frame is initialized and reconnected to this part. When this part's document opens, or as each display frame of this part becomes visible through scrolling or resizing, OpenDoc calls this method when it or this part's containing part instantiates and connects one of this part's previously stored display frames.

Your part's `DisplayFrameConnected` method should update your part's internal list of display frames and other related structures to reflect the connection of the specified display frame. If the specified frame is already in the list, no action is taken. Your part should also update relevant information in the display frame, for example, its used shape and presentation.

Your part's `DisplayFrameConnected` method is typically called after OpenDoc or this part's containing part calls a draft's `GetFrame` method. By contrast, a part's `DisplayFrameAdded` method is called after OpenDoc or this part's containing part calls a draft's `CreateFrame` method.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODPart::DisplayFrameAdded` method (page 478).

DisplayFrameRemoved

The `DisplayFrameRemoved` method should remove the specified frame from this part's internal list of display frames.

```
void DisplayFrameRemoved (in ODFrame frame);
```

`frame` A reference to a display frame to be removed.

DISCUSSION

To permanently delete one of its embedded frames, this part's containing part calls the embedded frame's `Remove` method. After removing the embedded frame, OpenDoc calls this method to notify this part of the removal.

Your part's `DisplayFrameRemoved` method should perform any actions necessary to remove a frame. For example, it should remove any frames (and part info) embedded within the specified frame. Your part's `DisplayFrameRemoved` method should update your part's internal list of display frames and other related structures to reflect the removal of the specified display frame. In addition, your method should call the arbitrator's `RelinquishFocusSet` method to relinquish any foci owned by the display frame being removed.

By contrast, OpenDoc calls a part's `DisplayFrameClosed` method to clean up when the part's document or window closes.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODPart::DisplayFrameClosed` method (page 479).

DisposeActionState

The `DisposeActionState` method should dispose of the undo action data.

```
void DisposeActionState (in ODActionData actionState,
                         in ODDoneState doneState);
```

`actionState`
A byte array whose buffer contains the data previously logged by this part to allow it to undo the action.

`doneState` The state of the undo action. The value of `doneState` must be one of the following: `kODDone`, `kODReDone`, or `kODUndone`.

DISCUSSION

The values `kODDone` and `kODReDone` for the `doneState` parameter indicate that the action was on the document's undo stack. The value `kODUndone` indicates that the action was on the document's redo stack.

OpenDoc calls this method when it disposes of the undo action data. After this method executes successfully, memory for the action state is reclaimed and is no longer usable for undo operations, effectively committing the action.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotUndo`	The specified action is not an undo action of this part.

SEE ALSO

The `ODActionData` type (page 868).
The `ODDoneState` type (page 869).
The `ODPart::ReadActionState` method (page 516).
The `ODPart::WriteActionState` method (page 531).

DragEnter

The `DragEnter` method should begin tracking a drag operation.

```
ODDragResult DragEnter (in ODDragItemIterator dragInfo,
                        in ODFacet facet,
                        in ODPoint where);
```

`dragInfo`	A reference to a drag-item iterator that describes the content, as well as the types and values, of the dragged data.
`facet`	A reference to a facet where the drag point is located.
`where`	The location of the drag point, expressed in frame coordinates.
return value	`kODTrue` if this part accepts drag-and-drop events in its display frame, otherwise `kODFalse`.

DISCUSSION

OpenDoc calls this method when the drag point has entered the specified facet of this part during a drag operation. Before calling this method, OpenDoc calls the `IsDroppable` method of the facet's frame to ensure that this part is able to receive a drop.

Your part's `DragEnter` method should examine the part kinds of the dragged data (using the drag-item iterator specified by the `dragInfo` parameter), and determine whether it can accept the dragged data. Your part should not at this

point attempt to read data from any of the storage units supplied by the drag-item iterator. If your part can accept a drop, it should provide the appropriate feedback to the user, such as displaying its drag border or changing the cursor appearance. If your part cannot handle the part kinds of the dragged data, it should take no action. If your part's `DragEnter` method returns `kODFalse`, OpenDoc assumes your part cannot accept a drop. OpenDoc will not make additional calls to your `DragWithin` or `Drop` methods as long as the mouse pointer remains in this facet.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotDrop`	This part does not support drag and drop.

SEE ALSO

The `ODDragResult` type (page 892).
The `ODPoint` type (page 855).
The `ODPart::DragLeave` method (page 484).
The `ODPart::DragWithin` method (page 485).

DragLeave

The `DragLeave` method should finish tracking a drag operation and deactivate this part from drag tracking.

```
void DragLeave (in ODFacet facet,
                in ODPoint where);
```

`facet`	A reference to a facet where the drag point is located.
`where`	The location of the drag point, expressed in frame coordinates.

DISCUSSION

OpenDoc calls this method when the drag point has left the specified facet of this part during a drag operation. After this method executes successfully, the part is guaranteed not to receive `DragWithin` or `Drop` messages until you receive another `DragEnter` message.

Your part's `DragLeave` method should clean up after a drag operation. For example, you should remove the drag border on the frame, remove highlighting from any content highlighted during drag tracking, and change the cursor appearance back to its original form.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotDrop`	This part does not support drag and drop.

SEE ALSO

The `ODPoint` type (page 855).
The `ODPart::DragEnter` method (page 483).
The `ODPart::DragWithin` method (page 485).

DragWithin

The `DragWithin` method should track a drag operation and provide graphical feedback regarding possible drop targets.

```
ODDragResult DragWithin (in ODDragItemIterator dragInfo,
                         in ODFacet facet,
                         in ODPoint where);
```

`dragInfo`	A reference to a drag-item iterator that describes the content, as well as the types and values, of the dragged data.
`facet`	A reference to a facet where the drag point is located.
`where`	The location of the drag point, expressed in frame coordinates.
return value	`kODTrue` if this part accepts drag-and-drop events in its display frame, otherwise `kODFalse`.

DISCUSSION

Your part's `DragWithin` method allows your part to do any desired processing inside of its display frame. For example, if your part allows objects to be dropped only on individual hot spots, it may change its feedback based on mouse-pointer location. If the mouse is not over these hot spots, the cursor may need to be changed to reflect that no dropping can be done, even though the mouse is in a droppable frame. OpenDoc calls this method continuously when the drag point is within the borders of the specified facet's frame.

Your part's `DragWithin` method should examine the part kinds of the dragged data (using the drag-item iterator specified by the `dragInfo` parameter) and determine whether it can accept the dragged data. Your part should not at this point attempt to read data from any of the storage units supplied by the drag-item iterator. If your part can accept a drop, it should provide the appropriate feedback to the user, for example, by displaying its drag border or changing the cursor appearance. If your part cannot handle the part kinds of the dragged data, it should take no action. If your part's `DragWithin` method returns `kODFalse`, OpenDoc assumes you no longer wish to accept a drop in this facet. It will not make additional calls to your `DragWithin` or `Drop` methods as long as the mouse pointer remains in this facet.

Your part's `DragWithin` method also represents an opportunity for your part to examine the state of the machine. For example, some parts may want to know whether the modifier keys are down while a drag operation is in progress.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotDrop`	This part does not support drag and drop.

SEE ALSO

The `ODDragResult` type (page 892).
The `ODPoint` type (page 855).
The `ODPart::DragEnter` method (page 483).
The `ODPart::DragLeave` method (page 484).

Draw

The `Draw` method should draw this part within the area that needs updating in the specified facet.

```
void Draw (in ODFacet facet,
           in ODShape invalidShape);
```

`facet` A reference to a facet in which this part is to draw.

`invalidShape`
A reference to a shape object defining the area of the facet that needs updating, expressed in frame coordinates.

DISCUSSION

OpenDoc calls this method when an update event occurs that involves a facet of this part. Your part's `Draw` method should make the actual platform-specific drawing calls.

Your part's `Draw` method should draw this part's content on the facet's canvas, updating the portion of the facet specified in the invalid shape. Your part must determine whether it should draw on the screen or to a printer, and then draw itself appropriately.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFacet::Draw` method (page 238).
The `ODFacet::Update` method (page 251).
The `ODWindow::Update` method (page 810).

Drop

The `Drop` method should move or copy the dragged data into this part.

```
ODDropResult Drop (in ODDragItemIterator dropInfo,
                   in ODFacet facet,
                   in ODPoint where);
```

`dropInfo`	A reference to a drag-item iterator that describes the content, as well as the types and values, of the dragged data.
`facet`	A reference to a facet where the drop has occurred.
`where`	The location of the drop point, expressed in frame coordinates.

return value The result of the drop operation. The return value is one of the following: `kODDropFail`, `kODDropCopy`, `kODDropMove`, or `kODDropUnfinished`.

DISCUSSION

The return value `kODDropFail` indicates an unsuccessful synchronous drop. On platforms that support asynchronous drag-and-drop operations, the return value `kODDropUnfinished` indicates that an asynchronous drop is in progress. The return value `kODDropCopy` indicates a successful synchronous drop with copy semantics. The return value `kODDropMove` indicates a successful synchronous drop with move semantics. These copy and move semantics are determined by examining the drag attributes and determining whether to display the Paste As dialog box. If a link is created, then your part's `Drop` method should return `kODDropCopy`, regardless of the drag attributes.

OpenDoc calls this method when the mouse button is released while the drag point is within a facet that can accept a drop.

Your part's `Drop` method should examine the part kinds of the dragged data (using the drag-item iterator specified by the `dropInfo` parameter) and determine whether it can accept the dragged data. If your part can accept a drop, this method should incorporate or embed the dropped data and return an appropriate return value indicating the result of the drop operation. If your part cannot accept a drop, this method should return `kODDropFail`.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotDrop`	This part does not support drag and drop.

SEE ALSO

The `ODDropResult` type (page 892).
The `ODPoint` type (page 855).

The `ODDragAndDrop::GetDragAttributes` method (page 187).
The `ODDragAndDrop::ShowPasteAsDialog` method (page 189).
The `ODPart::DropCompleted` method (page 490).

DropCompleted

The `DropCompleted` method is called to notify this part that a drop operation, resulting from an asynchronous drag initiated from this part, is complete.

```
void DropCompleted (in ODPart destPart,
                    in ODDropResult dropResult);
```

`destPart` A reference to a part where the drop is to be completed.

`dropResult` The result of the drop operation. The value of `dropResult` must be one of the following: `kODDropFail`, `kODDropCopy`, `kODDropMove`, or `kODDropUnfinished`.

DISCUSSION

The return value `kODDropFail` indicates an unsuccessful synchronous drop. The return value `kODDropCopy` indicates a successful synchronous drop with copy semantics. The return value `kODDropMove` indicates a successful synchronous drop with move semantics. On platforms that support asynchronous drag-and-drop operations, the return value `kODDropUnfinished` indicates that an asynchronous drop is in progress.

OpenDoc calls this method at the end of a drop operation resulting from an asynchronous drag initiated from this part.

If the drag-and-drop object's `StartDrag` method is executed synchronously, then that method's return value is this method's drop result.

Your part's `DropCompleted` method should perform any tasks that your part normally performs when it receives the result of a `StartDrag` method, modifying or restoring this part's data based on the drop result.

Mac OS
Asynchronous dragging is not supported for version 1.0 of OpenDoc for the Mac OS. This method is provided here for compatibility with future versions of OpenDoc. ◆

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotDrop`	This part does not support drag and drop.

SEE ALSO

The `ODDropResult` type (page 892).
The `ODDragAndDrop::StartDrag` method (page 192).
The `ODPart::Drop` method (page 488).

EditInLinkAttempted

The `EditInLinkAttempted` method should return a Boolean value that indicates whether this part maintains the destination of a link that includes the embedded frame.

```
ODBoolean EditInLinkAttempted (in ODFrame frame);
```

`frame`	A reference to an embedded frame, in which the edit was attempted, of this part.
return value	`kODTrue` if this part maintains the destination of a link that includes the embedded frame, otherwise `kODFalse`.

DISCUSSION

OpenDoc calls this method to notify this part that an attempt was made to edit content within a frame embedded in the destination of a link maintained by this part. The frame may be embedded at any depth within this part or its embedded parts.

Without making itself active, your part should display an alert box informing the user of the attempted edit to linked content and allow the user to find the source of the link or to break the link. If the user chooses to break the link, your part should change the link status of the embedded frame; the user can then retry the editing operation in the still-active frame.

Your part's `EditInLinkAttempted` method is not called by most parts. If your part's active frame is within a link destination and the user attempts to edit its content, you call the frame's `EditInLink` method instead of this method.

OVERRIDING

When you subclass `ODPart`, you should override this method if you support linking and embedding. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts that support linking.

EXCEPTIONS

`kODErrDoesNotLink`	This part does not support linking.

SEE ALSO

The `ODFrame::EditInLink` method (page 311).

EmbeddedFrameSpec

The `EmbeddedFrameSpec` method should create an object specifier for the specified embedded frame in this part.

```
void EmbeddedFrameSpec (in ODFrame embeddedFrame,
                        in ODObjectSpec spec);
```

`embeddedFrame`
A reference to an embedded frame of this part.

`spec` The object specifier for the frame.

DISCUSSION

An object specifier is a designation of a content object within a part. An object specifier is used to determine the target of a semantic event (a message sent to a part or one of its content elements). Object specifiers can be names ("blue rectangle") or logical designations ("word 1 of line 2 of embedded frame 3"). OpenDoc or parts can call this method to create an object specifier for an embedded frame to distinguish it from other frames.

If your part is an embedded part, this method should first obtain the specifier for your part's display frame by calling the `EmbeddedFrameSpec` method of your part's containing part and then concatenate that returned specifier with the object specifier for the specified embedded frame.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts that support scripting.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.
`kODErrInvalidFrame`	The specified frame is not an embedded frame of this part.

EmbeddedFrameUpdated

The `EmbeddedFrameUpdated` method should update any of this part's link-source objects affected by a change to the specified embedded frame.

```
void EmbeddedFrameUpdated (in ODFrame frame,
                           in ODUpdateID change);
```

`frame`	A reference to an embedded frame whose content has changed.
`change`	The update ID associated with the frame.

DISCUSSION

An embedded frame's `ContentUpdated` method calls this method when its content changes. Your part's `EmbeddedFrameUpdated` method is called recursively for all containing parts in the frame hierarchy through the root part of the window displaying the embedded frame whose content has changed. Therefore, this part is not responsible for notifying its containing part of the change. This part may ignore the notification if it is uninterested in changes to embedded content.

If the embedded frame is involved in a link source with your part, your part's `EmbeddedFrameUpdated` method should update the link-source object with the new data.

Your part's `EmbeddedFrameUpdated` method may be called multiple times with the same update ID. However, your part should wait a certain length of time (perhaps, a second) before updating its display so that subsequent calls to your method (with the same update ID) do not result in multiple updates for the same change.

OVERRIDING

When you subclass `ODPart`, you can override this method if you want your part to do something special in response to the `EmbeddedFrameUpdated` notification. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed` This part does not support embedding.

SEE ALSO

The `ODUpdateID` type (page 887).
The `ODFrame::ContentUpdated` method (page 308).

ExternalizeKinds

The `ExternalizeKinds` method should write to storage a representation for each part kind within the specified kind list that this part supports.

```
void ExternalizeKinds (in ODTypeList kindset);
```

`kindset` A reference to a type list specifying a set of part kinds.

DISCUSSION

OpenDoc calls this method. OpenDoc does not ensure that your part supports a subset of the part kinds in the specified kind list; you should write a representation for as many of the part kinds in the specified kind list that your part supports.

A part's `ExternalizeKinds` method does not specify anything about the ordering of those kinds in the contents property of your part. Make sure that the ordering of the values in your contents property reflects your part editor's fidelity order.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

FacetAdded

The `FacetAdded` method is called to notify this part that a facet has been added to one of its display frames.

```
void FacetAdded (in ODFacet facet);
```

`facet` A reference to a facet that has been added.

DISCUSSION

OpenDoc calls this method when your part's containing part adds a facet to one of your part's display frames. If your part is the root part in a window, OpenDoc calls this method when the window is opened.

Your part's `FacetAdded` method should perform any actions necessary to handle the addition of the new facet to one of your part's display frames. Some actions depend on the nature and implementation of your part itself, but others are standard. Standard actions include creating facets for all visible embedded frames within the area of the added facet, storing appropriate part info data in the facet, examining the facet's canvas to make sure your part editor understands how to draw on that canvas, and creating an offscreen canvas, if the facet needs one.

If your part does not support asynchronous display and is not a container part, it is not necessary for it to do anything except possibly validate that it can draw to its canvas.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFrame::FacetAdded` method (page 312).
The `ODPart::FacetRemoved` method (page 497).

FacetRemoved

The `FacetRemoved` method is called to notify this part that a facet has been removed from one of its display frames.

```
void FacetRemoved (in ODFacet facet);
```

`facet` A reference to a facet that has been removed.

DISCUSSION

OpenDoc calls this method when your part's containing part removes a facet from one of your part's display frames. If your part is the root part in a window, OpenDoc also calls this method when the window is closed.

Your part's `FacetRemoved` method should perform any actions necessary to handle the removal of the facet. In general, the actions performed by this method should reverse those performed by the `FacetAdded` method. If your part does not support asynchronous display and is not a container part, its response to this call can be minimal.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFacet` The specified facet is not a facet of this part.

SEE ALSO

The `ODFrame::FacetRemoved` method (page 313).
The `ODPart::FacetAdded` method (page 496).

FocusAcquired

The `FocusAcquired` method is called to notify this part that one of its display frames has acquired the specified focus.

```
void FocusAcquired (in ODTypeToken focus,
                    in ODFrame ownerFrame);
```

`focus` A tokenized string representing the focus type to be acquired, expressed as a 32-bit value.

`ownerFrame` A reference to the display frame that has acquired the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method to notify this part that the ownership of a focus has been transferred to it. For example, if a containing part uses the arbitrator's `TransferFocus` method to transfer a focus directly from one embedded part to another, the focus module calls the destination part's `FocusAcquired` method.

Your part's `FocusAcquired` method is not called when this part requests a focus or focus set using the arbitrator's `RequestFocus` and `RequestFocusSet` methods.

Your part's `FocusAcquired` method should perform any actions necessary to indicate that it has acquired the focus. For example, acquiring the selection focus might cause your part to highlight its selection.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODPart::FocusLost` method (page 499).
The `ODSession::Tokenize` method (page 598).

FocusLost

The `FocusLost` method is called to notify this part that one of its display frames has lost the specified focus.

```
void FocusLost (in ODTypeToken focus,
                in ODFrame ownerFrame);
```

`focus`	A tokenized string representing the focus type that was lost, expressed as a 32-bit value.
`ownerFrame`	A reference to a display frame that has lost the focus.

DISCUSSION

The `focus` parameter must be the tokenized form of one of the focus constants (`kODClipboardFocus`, `kODKeyFocus`, `kODMenuFocus`, `kODModalFocus`, `kODMouseFocus`, `kODScrollingFocus`, or `kODSelectionFocus`) or the tokenized form of a part-specific focus type. You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

OpenDoc calls this method to notify this part that the ownership of a focus has been unilaterally removed. For example, a focus module might detect that some physical hardware connection has been broken. Or if a containing part uses the arbitrator's `TransferFocus` method to transfer a focus directly from one embedded part to another, the focus module calls the source part's `FocusLost` method.

Your part's `FocusLost` method is not called when this part loses a focus because another part has requested it (that is, using the arbitrator's `RequestFocus` and `RequestFocusSet` methods). In this case, the part's `CommitRelinquishFocus` method is called if the part agrees to relinquish the focus.

Your part's `FocusLost` method should perform any actions necessary to indicate that it has lost the focus (for example, closing connections and removing highlighting). Your part should avoid inappropriate behavior after it has lost the focus. For example, your part should not attempt to adjust menus or display a menu bar when your part does not have the menu focus.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFocusType` type (page 859).
The `ODTypeToken` type (page 847).
The `ODPart::AbortRelinquishFocus` method (page 461).
The `ODPart::BeginRelinquishFocus` method (page 467).
The `ODPart::CommitRelinquishFocus` method (page 472).
The `ODPart::FocusAcquired` method (page 498).
The `ODSession::Tokenize` method (page 598).

FrameShapeChanged

The `FrameShapeChanged` method is called to notify this part that the frame shape of one of its display frames has changed.

```
void FrameShapeChanged (in ODFrame frame);
```

`frame` A reference to a display frame that was reshaped.

DISCUSSION

OpenDoc calls this method when it or this part's containing part changes the frame shape of one of this part's display frames.

This part should perform any actions necessary to respond to the new shape. For example, your part should change its content layout, change its used shape, or resize its embedded frames.

Your part also has the option of requesting a different frame shape using its display frame's `RequestFrameShape` method, though it must be able to accept the shape it is given. If the size of the frame is insufficient, your part may ask its containing part for additional frames by calling its containing part's `RequestEmbeddedFrame` method.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFrame::ChangeFrameShape` method (page 300).
The `ODPart::RequestEmbeddedFrame` method (page 521).
The `ODPart::RequestFrameShape` method (page 523).

FulfillPromise

The `FulfillPromise` method should fulfill a promise by providing the content data the promise represents.

```
void FulfillPromise (in ODStorageUnitView promiseSUView);
```

`promiseSUView`
A reference to a storage-unit view object that contains the promise. This is the same value created by the storage unit's `SetPromiseValue` method.

DISCUSSION

A promise is a specification of data to be transferred at a future time. For data interchange using drag and drop or the clipboard, the part transferring data should usually delay the actual data transfer, instead providing a promise that is fulfilled only when the transfer is ultimately required.

If a data transfer involves a very large amount of data, your part can choose to put out a promise instead of actually writing the data to a storage unit. Your `FulfillPromise` method can then write the actual data only if and when a transfer to another part occurs. When cloning in the `FulfillPromise` method, the clone kind should be the same as the one used when the promise was written. OpenDoc calls this method when a promise must be fulfilled.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrNoPromises`	This part did not make the promise.

SEE ALSO

The `ODStorageUnit::SetPromiseValue` method (page 686).

GeometryChanged

The `GeometryChanged` method is called to notify this part that the clip shape or external transform (or both) of one of this part's facets has changed.

```
void GeometryChanged (
                  in ODFacet facet,
                  in ODBoolean clipShapeChanged,
                  in ODBoolean externalTransformChanged);
```

`facet` A reference to a facet with the changed geometry.

`clipShapeChanged`
`kODTrue` if the clip shape has changed, otherwise `kODFalse`.

`externalTransformChanged`
`kODTrue` if the external transform has changed, otherwise `kODFalse`.

DISCUSSION

OpenDoc calls this method when this part's facet changes its clip shape, external transform, or both, or when this part's facet is repositioned or clipped differently.

Your part's `GeometryChanged` method should use the new clip shape for display. If your part only displays in response to update events, it does not need to do anything but check the clip shape each time it draws. If your part supports asynchronous display (for example, clocks and movies), it must notice the new clip shape and limit its display accordingly.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFacet`	The specified facet is not a facet of this part.

SEE ALSO

The `ODFacet::ChangeGeometry` method (page 230).

GetPrintResolution

The `GetPrintResolution` method should return the minimum desired resolution, in dots per inch, required for printing the content of the specified display frame.

```
ODULong GetPrintResolution (in ODFrame frame);
```

`frame`	A reference to a display frame for which the resolution is needed.
return value	The minimum desired resolution, expressed in dots per inch.

DISCUSSION

The root part calls this method when creating the document's print job to guarantee that it can display the highest-resolution part.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

GetRealPart

The `GetRealPart` method returns a reference to a part object encapsulated by the part wrapper.

```
ODPart GetRealPart ();
```

return value A reference to a part object encapsulated by the part wrapper.

DISCUSSION

Use of this method must conform to the following constraints:

- Only one client at a time may have access to the actual part. That is, clients should not hold on to a reference to the actual part object for longer than absolutely necessary; never store a reference to the part object and do not pass it as a parameter to other objects.
- Your part's `GetRealPart` method increments the reference count of the returned part. When you have finished using the actual part, you should call its `ReleaseRealPart` method. If another client attempts to get the actual part before the `ReleaseRealPart` method is called, an exception is generated.

Your part's `GetRealPart` method is almost never called unless you really need to access the actual part.

OVERRIDING

When you subclass `ODPart`, you must not override this method.

EXCEPTIONS

`kODErrPartNotWrapper`	This method was called on an actual part, not a part wrapper.

SEE ALSO

The `ODPart::IsRealPart` method (page 511).
The `ODPart::ReleaseRealPart` method (page 519).

HandleEvent

The `HandleEvent` method should attempt to handle the specified user event.

```
ODBoolean HandleEvent (inout ODEventData event,
                       in ODFrame frame,
                       in ODFacet facet,
                       inout ODEventInfo eventInfo);
```

`event`	A platform-specific structure representing an event. On return, the fields of the structure may have been modified. On the Mac OS platform, the structure is defined as a Mac OS event record.
`frame`	A reference to a display frame in which the event occurred.
`facet`	A reference to a facet in which the event occurred, or `kODNULL` for events not based on geometry (for example, keyboard events) or events outside a modal focus.
`eventInfo`	A platform-specific structure that contains additional event information. On return, the relevant fields of the structure are filled in if the event was handled.
return value	`kODTrue` if this part handled the event, otherwise `kODFalse`.

DISCUSSION

OpenDoc calls this method to pass user events to this part.

The `eventInfo` structure contains the following information:

- a reference to an embedded frame and an embedded facet of this part (for event types `kODEvtMouseDownEmbedded`, `kODEvtMouseUpEmbedded`, or `kODEvtMouseDownBorder`)
- an `ODPoint` value describing the location of the event, expressed in frame coordinates
- a Boolean value that indicates whether an embedded part propagated an event to this part

If this part has set the `doesPropagateEvents` flag for any of its embedded frames (by calling the embedded frame's `SetPropagateEvents` method), this part then receives any event not handled by an embedded frame in addition to it own events.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFacet`	The specified facet is not a facet of this part.
`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODEventData` type (page 860).
The `ODEventInfo` type (page 861).
The `ODFrame::SetPropagateEvents` method (page 329).

HighlightChanged

The `HighlightChanged` method should update the highlight state of the specified facet of this part.

```
void HighlightChanged (in ODFacet facet);
```

`facet` A reference to a facet of this part.

DISCUSSION

OpenDoc calls this method when the highlight state of one of your part's facets changes. This allows your part to draw its content consistently with the content highlighting of your part's containing part.

Your part's `HighlightChanged` method should adjust your part's presentation in the facet to its new highlight state. The new state is found by calling the facet's `GetHighlight` method.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFacet` The specified facet is not a facet of this part.

SEE ALSO

The `ODFacet::ChangeHighlight` method (page 231).
The `ODFacet::GetHighlight` method (page 243).

InitPart

The `InitPart` method initializes this part object.

```
void InitPart (in ODStorageUnit storageUnit,
               in ODPart partWrapper);
```

`storageUnit`
A reference to an empty storage unit to be used by this part as its primary persistent storage.

`partWrapper`
A reference to a part wrapper representing this part.

DISCUSSION

OpenDoc calls this method only once for the persistent lifetime of this part, when it is first created and has no stored data to read. After this method executes successfully, this part object is initialized and ready for use.

Your part's `InitPart` method, rather than its `somInit` method, should handle any initialization code that can potentially fail. Your part's `InitPart` method may attempt to allocate memory for your part instance, get resources that your part might need, or set up your part's persistent storage.

The inherited `InitPart` method creates and stores the `kODPropCreateDate`, `kODPropModDate`, and `kODPropModUser` properties; your part's storage unit automatically maintains the `kODPropModDate` and `kODPropModUser` properties.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must call its inherited method at the beginning of your implementation.

InitPartFromStorage

The `InitPartFromStorage` method initializes this part object from its stored data.

```
void InitPartFromStorage (in ODStorageUnit storageUnit,
                          in ODPart partWrapper);
```

`storageUnit`
A reference to a specified storage unit from which this part should read its persistent state.

`partWrapper`
A reference to a part wrapper representing this part.

DISCUSSION

Whenever a document containing this part is opened, or if this part is added to a document via data transfer, the part must be instantiated and read into memory. OpenDoc calls this method to initialize the runtime part object from persistent storage.

Your part's `InitPartFromStorage` method is similar to its `InitPart` method, except that it reads in data. OpenDoc passes a storage unit to your part, from which your part reads itself. Your part is responsible for reading the storage unit and preparing itself to receive other messages. Your part should retrieve, from its contents property, the value that represents the data stream to be read.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must call its inherited method at the beginning of your implementation.

IsRealPart

The `IsRealPart` method returns a Boolean value that indicates whether this part is an actual part (as opposed to a part wrapper).

```
ODBoolean IsRealPart ();
```

return value `kODTrue` if this part is an actual part, or `kODFalse` if this part is a part wrapper.

DISCUSSION

Your part's `IsRealPart` method is almost never called unless you really need to access the actual part.

OVERRIDING

When you subclass `ODPart`, you must not override this method.

SEE ALSO

The `ODPart::GetRealPart` method (page 505).
The `ODPart::ReleaseRealPart` method (page 519).

LinkStatusChanged

The `LinkStatusChanged` method is called to notify this part that the link status of one of its display frames has changed.

```
void LinkStatusChanged (in ODFrame frame);
```

`frame` A reference to a display frame whose link status has changed.

DISCUSSION

OpenDoc calls this method. Your part's `LinkStatusChanged` method should call the `ChangeLinkStatus` method of each embedded frame affected by the display frame change, assigning it the same link status as the display frame.

If the value `kODNotInLink` was assigned in the `ChangeLinkStatus` method, then your part's `LinkStatusChanged` method can still get the status of this part's display frame (its containing frame).

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODFrame::ChangeLinkStatus` method (page 302).

LinkUpdated

The `LinkUpdated` method should replace the content at each destination of a link with new content from an updated link object.

```
void LinkUpdated (in ODLink updatedLink,
                  in ODUpdateID change);
```

`updatedLink`
A reference to a link that has changed.

`change` The update ID associated with the link; an identifier for a particular version of link-source data.

DISCUSSION

Each link object maintains a registry of dependent parts. If this part is registered as a dependent of the link source (by having called the link's

`RegisterDependent` method), OpenDoc calls this method automatically when the link destination object changes.

Your part's `LinkUpdated` method should retrieve the data from the link and incorporate or embed that data into your part at the link's destination, thereby replacing any previous content of the link.

It is not always necessary to update link destinations immediately. For example, if the destination has scrolled offscreen but is registered as a dependent of the link, updating does not need to occur until the destination scrolls back into view. Your part editor can, if desired, perform link updating as a background task.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotLink`	The specified link is not a link of this part.

SEE ALSO

The `ODUpdateID` type (page 887).
The `ODLink::RegisterDependent` method (page 346).

Open

The `Open` method should create or activate a window in which a frame of this part is the root frame.

```
ODID Open (in ODFrame frame);
```

`frame`	A reference to a frame that is being opened into a window, or `kODNULL` if the frame does not exist.

return value The ID associated with the window.

DISCUSSION

Your part is always responsible for creating windows in which it is the root part, even when OpenDoc is opening a saved draft. Your part's `Open` method is called in these circumstances:

- When this part is initially created from stationery—meaning that it has no previously stored frame or window information—OpenDoc calls this method and passes `kODNULL` for the `frame` parameter.
- When this part is an embedded part whose frame is selected, and the user chooses the Open Selection command from the Document menu, this part's containing part calls this part's `Open` method and passes a reference to the selected frame in the `frame` parameter.
- When this part is the root part of a document being opened, OpenDoc calls this method and passes a reference to the root frame in the `frame` parameter. The root frame is annotated with a storage unit containing window information; if the root frame has no window information, this part should instead open a default window.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODID` type (page 869).
The `ODWindowState::Internalize` method (page 833).
For more information on opening windows, see the chapter on windows and menus in the *OpenDoc Programmer's Guide for the Mac OS*.

PresentationChanged

The `PresentationChanged` method is called to notify this part that the presentation of one of its display frames has changed.

```
void PresentationChanged (in ODFrame frame);
```

`frame` A reference to a display frame for this part.

DISCUSSION

OpenDoc calls this method when this part's display frame changes its presentation.

Your part's `PresentationChanged` method should examine the new presentation using its frame's `GetPresentation` method. It should then display itself in the specified display frame according to the indicated presentation. If your part does not support the requested presentation, it should instead pick a presentation that it can support and then call its frame's `SetPresentation` method to update the presentation in the frame.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFrame::ChangePresentation` method (page 304).
The `ODFrame::SetPresentation` method (page 328).

ReadActionState

The `ReadActionState` method should read the undo action data from the specified storage-unit view object.

```
ODActionData ReadActionState (
                        in ODStorageUnitView storageUnitView);
```

`storageUnitView`
A reference to a storage-unit view object that contains the action data.

return value A byte array whose buffer contains the data previously logged by this part to allow it to undo the action.

DISCUSSION

OpenDoc calls this method when it reads the undo action data from persistent storage. Your part's `ReadActionState` method will not be called in OpenDoc 1.0 due to the lack of a persistent undo model; it is provided here for compatibility with future versions of OpenDoc.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotUndo`	The specified action is not an undo action of this part.
`kODErrOutOfMemory`	There is not enough memory to read the data.

SEE ALSO

The `ODActionData` type (page 868).
The `ODPart::DisposeActionState` method (page 482).
The `ODPart::WriteActionState` method (page 531).

ReadPartInfo

The `ReadPartInfo` method should read the part info data for a display frame of this part from the specified storage-unit view object.

```
ODInfoType ReadPartInfo (
                    in ODFrame frame,
                    in ODStorageUnitView storageUnitView);
```

`frame` A reference to a display frame for this part.

`storageUnitView`
A reference to a storage-unit view object that is focused to the frame's part info property, but not to any value within that property.

return value The information stored in the frame's part info.

DISCUSSION

When a document is reopened, OpenDoc may ask this part to read a display frame's part info data from a particular storage unit back into memory. OpenDoc calls this method when it reads in the part info data for a display frame of this part.

Your part's `ReadPartInfo` method should get the storage unit associated with the specified storage-unit view and focus it to the value(s) in the frame's part info property as necessary to read in the formats it needs.

Your part's `ReadPartInfo` method should read the data from the storage unit and place it in a block of memory, if necessary. (Your part must first allocate the block of memory.) In simple cases, part info data could be simple data, such as an integer, which does not require a block of memory. Your part's

`ReadPartInfo` method should then return the memory block to the frame for storage so that your part can access it later.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.
`kODErrOutOfMemory`	There is not enough memory to allocate the part info data.

SEE ALSO

The `ODInfoType` type (page 853).
The `ODPart::ClonePartInfo` method (page 471).
The `ODPart::WritePartInfo` method (page 532).

RedoAction

The `RedoAction` method should redo the specified action.

```
void RedoAction (in ODActionData actionState);
```

`actionState`
A byte array whose buffer contains the data previously logged by this part to allow it to redo the action.

DISCUSSION

A part may need to give the user the capability of repeating the effects of recently undone commands. OpenDoc calls this method when the user chooses to redo an action of this part.

Your part is responsible for notifying the clipboard whenever a Cut, Copy, or Paste operation is done, undone, or redone. If your part's `RedoAction` method is called, it should remove the object from its content model and notify the clipboard that the cut was redone.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotUndo`	The specified action is not an undo action of this part.

SEE ALSO

The `ODActionData` type (page 868).
The `ODPart::UndoAction` method (page 528).

ReleaseRealPart

The `ReleaseRealPart` method releases the part object encapsulated by the part wrapper.

```
void ReleaseRealPart ();
```

DISCUSSION

For part wrappers, this method marks the part object as available for access by another client. Your part's `ReleaseRealPart` method is almost never called unless you really need to access the actual part.

OVERRIDING

When you subclass `ODPart`, you must not override this method.

EXCEPTIONS

`kODErrPartNotWrapper`	This method was called on an actual part, not a part wrapper.

SEE ALSO

The `ODPart::GetRealPart` method (page 505).
The `ODPart::IsRealPart` method (page 511).

RemoveEmbeddedFrame

The `RemoveEmbeddedFrame` method should remove the specified embedded frame from this part's content.

```
void RemoveEmbeddedFrame (in ODFrame embeddedFrame);
```

`embeddedFrame`
A reference to an embedded frame of this part.

DISCUSSION

An embedded part calls this part's `RemoveEmbeddedFrame` method when it no longer wants to display itself in the specified frame.

If your part supports embedding, your part's `RemoveEmbeddedFrame` method should respond to a request to remove an embedded frame. Your

part's `RemoveEmbeddedFrame` method should only remove frames that were added by calls to your part's `RequestEmbeddedFrame` method.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.
`kODErrInvalidFrame`	The specified frame is not an embedded frame of this part.

SEE ALSO

The `ODPart::RequestEmbeddedFrame` method (page 521).

RequestEmbeddedFrame

The `RequestEmbeddedFrame` method should create a new display frame for the specified embedded part.

```
ODFrame RequestEmbeddedFrame (in ODFrame containingFrame,
                              in ODFrame baseFrame,
                              in ODShape frameShape,
                              in ODPart embedPart,
                              in ODTypeToken viewType,
                              in ODTypeToken presentation,
                              in ODBoolean isOverlaid);
```

`containingFrame`
A reference to a display frame in which to embed the new frame.

`baseFrame`	A reference to a sibling frame of the frame to be created, and also a reference to a display frame for the embedded part.
`frameShape`	A reference to a requested shape for the new frame, expressed in the frame coordinates of the containing frame.
`embedPart`	A reference to a part that is to be displayed in the new frame.
`viewType`	A tokenized string representing the view type to assign to this part's frame.
`presentation`	A tokenized string representing the presentation to assign to this part's frame.
`isOverlaid`	`kODTrue` if this part's frame is to be an overlaid frame, otherwise `kODFalse`.
return value	A reference to a new frame object.

DISCUSSION

The `viewType` parameter must be the tokenized form of one of the view-type constants (`kODViewAsFrame`, `kODViewAsSmallIcon`, `kODViewAsLargeIcon`, or `kODViewAsThumbnail`). You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

An embedded part calls its containing part's `RequestEmbeddedFrame` method when it needs an extra frame to flow content into, for example, an additional column or additional page of text. The part must specify one of its current display frames as a base frame; the new frame is a sibling of the base frame, and it is in the same group as the base frame.

If your part supports embedding, it may need to respond to a request to add an embedded frame. Your part's `RequestEmbeddedFrame` method should call your draft's `CreateFrame` method to create the new frame and then return it to the embedded part.

Before returning the frame object, your part's `RequestEmbeddedFrame` method should call the frame object's `Acquire` method. When the caller has finished using the returned frame object, it should call the frame object's `Release` method.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.
`kODErrInvalidFrame`	The specified sibling frame is not an embedded frame of this part.
`kODErrOutOfMemory`	There is not enough memory to embed a part.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODDraft::CreateFrame` method (page 162).
The `ODSession::Tokenize` method (page 598).
For more information on frame negotiation, see the layout and embedding chapter in the *OpenDoc Programmer's Guide for the Mac OS*.

RequestFrameShape

The `RequestFrameShape` method is called to negotiate a new frame shape for the specified frame embedded in this part.

```
ODShape RequestFrameShape (in ODFrame embeddedFrame,
                           in ODShape frameShape);
```

`embeddedFrame`
A reference to an embedded frame for which the frame-shape change is requested.

`frameShape` A reference to a requested shape, expressed in the frame coordinates of the embedded frame.

return value A reference to a new shape for the embedded frame, expressed in frame coordinates.

DISCUSSION

OpenDoc calls this method to initiate a frame negotiation process requested by this part's embedded part.

Your part's `RequestFrameShape` method should decide what new shape to give an embedded frame, using the requested frame shape as a guideline, and return a reference to the shape object it allows the embedded frame to have. The embedded frame stores the shape as its new frame shape and returns the shape to its part so that its part knows what its new shape is. The embedded part must accept the returned shape, although it may make further requests for different shapes or additional frames.

Before returning the shape object, your part's `RequestFrameShape` method should call the shape object's `Acquire` method. When the caller has finished using the returned shape object, it should call the shape object's `Release` method.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed` This part does not support embedding.

SEE ALSO

The `ODFrame::RequestFrameShape` method (page 323).
For more information on frame negotiation, see the layout and embedding chapter in the *OpenDoc Programmer's Guide for the Mac OS*.

RevealFrame

The `RevealFrame` method should make the specified embedded frame visible by scrolling it into view.

```
ODBoolean RevealFrame (in ODFrame embeddedFrame,
                       in ODShape revealShape);
```

`embeddedFrame`
A reference to an embedded frame of this part.

`revealShape`
A reference to a shape object, expressed in frame coordinates, that indicates the portion of the frame to be revealed.

return value `kODTrue` if this part was able to reveal the frame, otherwise `kODFalse`.

DISCUSSION

An embedded part calls this part's `RevealFrame` method when it needs to become visible, such as in conjunction with a keyboard event that requires an undisplayed area of a window to scroll into view.

Your part's `RevealFrame` method should scroll one of your part's display frames, if necessary, to make the specified embedded frame visible. If your part has no visible display frames, it should ask its containing part to reveal one of them. If your part has no display frames, or if your part's containing frame cannot reveal the display frame, your method should open a frame in a new window.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.
`kODErrInvalidFrame`	The specified frame is not an embedded frame of this part.

SEE ALSO

The `ODFrame::AcquireContainingFrame` method (page 295).

RevealLink

The `RevealLink` method should show the content at the source of a link.

```
void RevealLink (in ODLinkSource linkSource);
```

`linkSource` A reference to a link-source object representing the linked content to be revealed.

DISCUSSION

OpenDoc calls this method when a link source needs to be shown; this method is not called by parts.

Your part's `RevealLink` method should select the linked content to be revealed in one of its frames, scroll the frame into view using its containing part's `RevealFrame` method, and make that frame active. If your part has no visible display frames, or if your part's containing frame cannot reveal the display frame, your method should open a frame in a new window.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotLink`	The specified link is not a link of this part.

SEE ALSO

The `ODPart::RevealFrame` method (page 525).

SequenceChanged

The `SequenceChanged` method is called to notify this part that the sequencing of this part's display frame within its frame group has changed.

```
void SequenceChanged (in ODFrame frame);
```

`frame`	A reference to a display frame whose sequence has been reordered.

DISCUSSION

OpenDoc calls this method when this part's containing part adds a new frame to the group or reorders the frames in the group. The containing part calls its embedded frame's `ChangeSequenceNumber` method to initiate the change.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFrame::ChangeSequenceNumber` method (page 305).

UndoAction

The `UndoAction` method should undo the specified action.

```
void UndoAction (in ODActionData actionState);
```

`actionState`
A byte array whose buffer contains the data previously logged by this part to allow it to undo the action.

DISCUSSION

OpenDoc calls this method when the user chooses to undo an action of this part.

Your part may need to give the user the capability of reversing the effects of recently executed commands. If it does, your part's `UndoAction` method should perform any reverse editing necessary to restore itself to the state it possessed before the specified action.

Your part is responsible for notifying the clipboard whenever a Cut, Copy, or Paste operation is done, undone, or redone. When a part cuts an object to the clipboard, a reference to the object should be saved in an undo action. If your part's `UndoAction` method is called, it should reinstate the object into its content model from its undo information and notify the clipboard that the cut was undone.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotUndo`	The specified action is not an undo action of this part.

SEE ALSO

The `ODActionData` type (page 868).
The `ODPart::RedoAction` method (page 518).

UsedShapeChanged

The `UsedShapeChanged` method is called to notify this part that the used shape of one of its embedded frames has changed.

```
void UsedShapeChanged (in ODFrame embeddedFrame);
```

`embeddedFrame`
A reference to an embedded frame of this part.

DISCUSSION

OpenDoc calls this method when a frame embedded in this part changes its used shape. Containing parts that have wrapped their content to the contour of an embedded frame's used shape (as in text wrapping) need to adjust the layout of that content for the new used shape.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely. This method needs to be implemented only by container parts.

EXCEPTIONS

`kODErrCannotEmbed`	This part does not support embedding.

`kODErrInvalidFrame`	The specified frame is not an embedded frame of this part.

SEE ALSO

The `ODFrame::ChangeUsedShape` method (page 305).

ViewTypeChanged

The `ViewTypeChanged` method is called to notify this part that the view type of one of its display frames has changed.

```
void ViewTypeChanged (in ODFrame frame);
```

`frame`	A reference to a display frame for this part.

DISCUSSION

OpenDoc calls this method when this part's display frame changes its view type.

Your part's `ViewTypeChanged` method should examine the new view type using its frame's `GetViewType` method. It should then display itself in the specified display frame according to the indicated view type. If your part does not support the requested view type, it should instead pick a view type that it can support and then call its frame's `SetViewType` method to update the view type in the frame.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrInvalidFrame`	The specified frame is not a display frame of this part.

SEE ALSO

The `ODFrame::GetViewType` method (page 316).
The `ODFrame::SetViewType` method (page 330).

WriteActionState

The `WriteActionState` method should write the undo action data into the specified storage-unit view object.

```
void WriteActionState (
                    in ODActionData actionState,
                    in ODStorageUnitView storageUnitView);
```

`actionState`
A byte array whose buffer contains the data previously logged by this part to allow it to redo the action.

`storageUnitView`
A reference to a storage-unit view object where the action data is to be written to storage.

DISCUSSION

OpenDoc calls this method when it writes the undo action data to persistent storage. Your part's `WriteActionState` method will not be called in OpenDoc 1.0 due to the lack of a persistent undo model; it is provided here for compatibility with future versions of OpenDoc.

OVERRIDING

When you subclass `ODPart`, you can override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

EXCEPTIONS

`kODErrDoesNotUndo`	The specified action is not an undo action of this part.

SEE ALSO

The `ODActionData` type (page 868).
The `ODPart::DisposeActionState` method (page 482).
The `ODPart::ReadActionState` method (page 516).

WritePartInfo

The `WritePartInfo` method should write the part info data for a display frame of this part into the specified storage-unit view object.

```
void WritePartInfo (in ODInfoType partInfo,
                    in ODStorageUnitView storageUnitView);
```

`partInfo` The part info data to write.

`storageUnitView`
A reference to a storage-unit view object that is focused to the frame's part info property, but not to any value within that property.

DISCUSSION

When a document is saved (using the Save command from the Document menu), OpenDoc may ask this part to save a display frame's part info data to a particular storage unit. OpenDoc calls this method when it writes out the part info data for a display frame of a part.

Your part's `WritePartInfo` method should get the storage unit associated with the specified storage-unit view and focus it to the value(s) in the frame's part info property as necessary to write the formats it needs.

OVERRIDING

When you subclass `ODPart`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

SEE ALSO

The `ODInfoType` type (page 853).
The `ODPart::ClonePartInfo` method (page 471).
The `ODPart::ReadPartInfo` method (page 517).

ODPersistentObject

Superclasses `ODRefCntObject` → `ODObject`

Subclasses `ODFrame`, `ODLink`, `ODLinkSource`, and `ODPart`

An object of the `ODPersistentObject` class implements the protocol for saving and restoring objects to persistent storage.

Description

Certain objects created in one OpenDoc session can be saved to persistent storage; in a later session, the same objects can be re-created from their stored data. Similarly, an object that is not currently being used can be saved to persistent storage to free space in memory; if the object is needed later in the same session, it can be re-created from its stored data. An object whose state can be saved to persistent storage is called a **persistent object**.

The `ODPersistentObject` class is the abstract superclass for all OpenDoc classes whose objects need to be saved persistently. You should not create immediate subclasses or instances of `ODPersistentObject` itself. Three of its subclasses are concrete classes: `ODFrame`, `ODLink`, and `ODLinkSource`. You can create instances of those classes, but you should not create subclasses of them. The `ODPersistentObject` class has a fourth subclass, `ODPart`, which is an abstract class. To develop a part editor, you need to create a subclass of `ODPart`. The persistent objects that can be saved and re-created from storage are frames, links, link sources, and parts.

Each persistent object has an associated storage unit in which it can store its data persistently. Within each draft in a particular session, the storage unit is given a unique identifier (ID) that designates both the storage unit itself and the persistent object whose data it contains. This ID is not part of the object's persistent data but is valid only for the duration of the session.

The process of storing a persistent object's data is called externalizing the object. When a persistent object writes itself to storage, it writes into its storage unit whatever information is necessary to restore it to its current state. When a

stored persistent object is re-created, it initializes itself to its previous state by reading the data it stored when it was last written.

A persistent object can also be **cloned**, that is, the object and all additional objects that it references can be copied. Typically, an object is cloned whenever its data must be transferred, for example, when it is copied to the clipboard.

For more information about storage units, see the `ODStorageUnit` class description (page 641). For more information about cloning, see the chapter on data transfer in the *OpenDoc Programmer's Guide for the Mac OS*. For more information about the different kinds of persistent objects, see the descriptions for the classes `ODFrame` (page 288), `ODLink` (page 339), `ODLinkSource` (page 361), and `ODPart` (page 445).

Methods

This section presents summary descriptions of the `ODPersistentObject` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Initializing and Releasing

`InitPersistentObject`
: Initializes this newly created persistent object.

`InitPersistentObjectFromStorage`
: Initializes this re-created persistent object from its stored data.

`ReleaseAll`
: Releases all transitory references from this persistent object to other reference-counted objects.

Accessing

`GetID`
: Returns the unique ID of this persistent object for the current draft in the current session.

`GetStorageUnit`
: Returns a reference to the storage unit in which this persistent object stores its data.

Writing

`Externalize`
: Stores the data needed to restore this persistent object to its current state.

`CloneInto`	Clones this persistent object by copying its data into the specified storage unit.

CloneInto

The `CloneInto` method clones this persistent object by copying its data into the specified storage unit.

```
void CloneInto (in ODDraftKey key,
                in ODStorageUnit toSU,
                in ODFrame scope);
```

`key`	The draft key identifying the current cloning operation, expressed as a 32-bit value. The key provides thread-safe access to cloning.
`toSU`	A reference to the destination storage unit to which the data is to be copied.
`scope`	A reference to the frame object that defines the scope of the cloning operation.

DISCUSSION

Your part should never call this method directly; it is called by the draft's `Clone` or `WeakClone` methods.

The `scope` parameter determines which of the referenced objects are within the scope of this cloning operation. Typically, the `scope` parameter is a reference to a frame and only those objects embedded in that frame are within the scope. In the rare case in which the `scope` parameter is `kODNULL`, all referenced objects are within the scope.

This method copies this object's data into the specified destination storage unit and clones any additional objects to which this object has strong and weak persistent references and that are within the scope of this cloning operation. Objects referenced by strong persistent references are strongly cloned by recursive calls to the `Clone` method; objects referenced by weak persistent references are weakly cloned by calls to the `WeakClone` method. Otherwise,

only those objects logically enclosed in the specified frame are within the scope and should be cloned.

OVERRIDING

Every subclass of `ODPersistentObject` must override the `CloneInto` method to support data transfer of internal data. The override method must call its inherited `CloneInto` method at the beginning of its implementation.

SEE ALSO

The `ODDraftKey` type (page 872).
The `ODDraft::Clone` method (page 159).
The `ODDraft::WeakClone` method (page 181).
The `ODPart` class (page 445).

Externalize

The `Externalize` method stores the data needed to restore this persistent object to its current state.

```
void Externalize ();
```

DISCUSSION

Your part calls this method of a persistent object (for example, a part or frame) to write that object's data to persistent storage. This method writes the object's persistent state to the properties and values in the object's storage unit. It must write sufficient information to allow the object's `Init`*Class*`FromStorage` method to restore the object to its current state; for example, a part's `Externalize` method must write the information needed by its `InitPartFromStorage` method.

OVERRIDING

Every subclass of `ODPersistentObject` must override the `Externalize` method. Your subclass of `ODPart` overrides this method if your part maintains persistent content. The override method must call its inherited `Externalize` method at the beginning of its implementation.

SEE ALSO

The `ODPersistentObject::InitPersistentObject` method (page 539).
The `ODPersistentObject::InitPersistentObjectFromStorage` method (page 540).
The `ODPart` class (page 445).

GetID

The `GetID` method returns the unique ID of this persistent object for the current draft in the current session.

```
ODID GetID ();
```

return value The unique ID of this persistent object.

DISCUSSION

The ID of the persistent object is also the ID of its storage unit. The ID is not persistent data. This persistent object may have a different ID in each different session and in each different draft within the same session.

SEE ALSO

The `ODID` type (page 869).

GetStorageUnit

The `GetStorageUnit` method returns a reference to the storage unit in which this persistent object stores its data.

```
ODStorageUnit GetStorageUnit ();
```

return value A reference to the storage unit of this persistent object, or `kODNULL` if the object does not have a storage unit.

DISCUSSION

Whenever your part calls this method, it should check that the return value is non-null before attempting to use the storage unit because nonpersistent frames, which do not have storage units, may be added to your part.

You should never cache a reference to the returned storage unit; instead you must call this method whenever you access the storage unit. This method does not increment the reference count of the returned storage unit.

InitPersistentObject

The `InitPersistentObject` method initializes this newly created persistent object.

```
void InitPersistentObject (in ODStorageUnit storageUnit);
```

`storageUnit`
A reference to a storage unit of this persistent object.

DISCUSSION

Your part editor should never call this method directly; it is called automatically whenever a persistent object is created for the first time. Every existing OpenDoc subclass of `ODPersistentObject` (including `ODPart`) has an initialization method that calls the inherited `InitPersistentObject` method at the beginning of its implementation.

For example, the initialization method of `ODPart` is the `InitPart` method. When you call the draft's `CreatePart` method to create a part of your class, that factory method calls your part's `InitPart` method. Your part's `InitPart` method should call the inherited `InitPart` method, which calls the `InitPersistentObject` method.

The `InitPersistentObject` method is not called when a stored object is re-created. Instead, the `InitPersistentObjectFromStorage` method is called to restore the object to its state at the time it was last saved.

SEE ALSO

The `ODPart::InitPart` method (page 509).
The `ODPersistentObject::Externalize` method (page 537).
The `ODPersistentObject::InitPersistentObjectFromStorage` method (page 540).

InitPersistentObjectFromStorage

The `InitPersistentObjectFromStorage` method initializes this re-created persistent object from its stored data.

```
void InitPersistentObjectFromStorage (
                              in ODStorageUnit storageUnit);
```

`storageUnit`
A reference to a storage unit of this persistent object.

DISCUSSION

Your part editor should never call this method directly; it is called automatically whenever a persistent object is re-created from stored data. Every existing OpenDoc subclass of `ODPersistentObject` (including `ODPart`) has an `Init`*Class*`FromStorage` method to initialize an object of that class from its stored data; that method calls the inherited `InitPersistentObjectFromStorage` method and then reads any data that was previously written by the `Externalize` method when the object was

stored. The object's factory method calls the object's `Init`*Class*`FromStorage` method when re-creating the object from its stored data.

For example, the `ODFrame` class has a private `InitFrameFromStorage` method. When you call the draft's `AcquireFrame` method to re-create a frame, that factory method calls the re-created frame's `InitFrameFromStorage` method.

SEE ALSO

The `ODPart::InitPartFromStorage` method (page 510).
The `ODPersistentObject::Externalize` method (page 537).
The `ODPersistentObject::InitPersistentObject` method (page 539).

ReleaseAll

The `ReleaseAll` method releases all transitory references from this persistent object to other reference-counted objects.

```
void ReleaseAll ();
```

DISCUSSION

The factory object that creates each persistent object calls that object's `ReleaseAll` method before deleting it.

OVERRIDING

Every subclass of `ODPersistentObject` overrides this method if its objects maintain references to other persistent objects. The override method must call its inherited `ReleaseAll` method at the end of its implementation.

SEE ALSO

The `ODPart` class (page 445).

ODPlatformTypeList

Superclasses `ODObject`

Subclasses none

An object of the `ODPlatformTypeList` class is an ordered set of `ODPlatformType` elements.

Description

A platform type list is an ordered set of elements, each specifying a different value of some platform-specific type. Because these elements are of the `ODPlatformType` type (page 889), they can represent any platform-specific value used to identify data formats for data interchange.

To create a platform type list, call the `CreatePlatformTypeList` method (page 638) of the storage-system object. If the call to that method specifies an existing platform type list, the new platform type list is initialized to a copy of that list; otherwise, the new list is initialized to an empty list (a list with no elements).

You can add elements one at a time to the end of the platform type list. OpenDoc ensures that each element of a platform type list is unique; if you attempt to add a value that is already in the list, the list remains unchanged. You can remove elements from the list, test whether the list contains a particular element, and get the number of elements in the list. If you need to perform an operation for each element of the list, you can create an object of the `ODPlatformTypeListIterator` class (page 547) and use it to iterate through the list.

Methods

This section presents summary descriptions of the `ODPlatformTypeList` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Manipulating Elements

`AddLast` Adds an element to the end of this platform type list.

`Remove` Removes the specified element from this platform type list.

Testing

`Contains` Returns a Boolean value that indicates whether this platform type list contains the specified element.

`Count` Returns the number of elements in this platform type list.

Creating an Iterator

`CreatePlatformTypeListIterator`
Creates a platform type-list iterator for this platform type list.

AddLast

The `AddLast` method adds an element to the end of this platform type list.

```
void AddLast (in ODPlatformType type);
```

`type` The element to be added to the list.

DISCUSSION

If this platform type list already contains the specified element, no action is taken. Otherwise, the specified element is added to the end of the list.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to add the specified element to this platform type list.

SEE ALSO

The `ODPlatformType` type (page 889).
The `ODPlatformTypeList::Contains` method (page 544).
The `ODPlatformTypeList::Remove` method (page 546).

Contains

The `Contains` method returns a Boolean value that indicates whether this platform type list contains the specified element.

```
ODBoolean Contains (in ODPlatformType type);
```

`type`	The element to be tested for inclusion in this list.
return value	`kODTrue` if this platform type list contains the specified element, otherwise `kODFalse`.

SEE ALSO

The `ODPlatformType` type (page 889).
The `ODPlatformTypeList::AddLast` method (page 543).
The `ODPlatformTypeList::Remove` method (page 546).

Count

The `Count` method returns the number of elements in this platform type list.

```
ODULong Count ();
```

return value The number of elements in this platform type list, expressed as an unsigned 32-bit value or 0 if the list is empty.

CreatePlatformTypeListIterator

The `CreatePlatformTypeListIterator` method creates a platform type-list iterator for this platform type list.

```
ODPlatformTypeListIterator
                    CreatePlatformTypeListIterator ();
```

return value A reference to the newly created platform type-list iterator.

DISCUSSION

You call this method if you need to apply an operation to each element of this platform type list.

While you are using the returned platform type-list iterator, you must not modify this platform type list; in particular, you must not add or remove elements, and you must not delete this list platform type list.

You must delete the returned platform type-list iterator when it is no longer needed.

EXCEPTIONS

`kODErrOutOfMemory` There is not enough memory to create the iterator.

SEE ALSO

The `ODPlatformTypeListIterator` class (page 547).

Remove

The `Remove` method removes the specified element from this platform type list.

```
void Remove (in ODPlatformType type);
```

`type` The element to be removed.

DISCUSSION

If this platform type list does not contain the specified element, no action is taken.

SEE ALSO

The `ODPlatformType` type (page 889).
The `ODPlatformTypeList::AddLast` method (page 543).
The `ODPlatformTypeList::Contains` method (page 544).

ODPlatformTypeListIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODPlatformTypeListIterator` class provides access to each element of a platform type list.

Description

You use a platform type-list iterator to apply an operation to each element of a platform type list.

Your part creates a platform type-list iterator object by calling the platform type list object's `CreatePlatformTypeListIterator` method (page 545), which returns a reference to a platform type-list iterator object. Platform type-list iterator objects are not currently required by any OpenDoc methods, but your part may wish to use these objects for internal use.

While you are using a platform type-list iterator, you should not modify or delete the platform type list that created it. You must postpone adding elements to or removing elements from the platform type list until after you have deleted the iterator.

For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODPlatformTypeListIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and returns the first element in the platform type list that created this platform type-list iterator.
`Next`	Returns the next element in the platform type list that created this platform type-list iterator.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and returns the first element in the platform type list that created this platform type-list iterator.

```
ODPlatformType First ();
```

return value The first element in the platform type list, or `kODNULL` if the platform type list is empty.

DISCUSSION

Your part must call this method before calling this platform type-list iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

EXCEPTIONS

`kODErrIteratorOutOfSync`
The platform type list was modified while the iteration was in progress.

SEE ALSO

The `ODPlatformType` type (page 889).

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more elements remain in the platform type list. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found an element. This method returns `kODFalse` when you have examined all the elements. If the platform type list that created this iterator is empty, this method always returns `kODFalse`.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The platform type list was modified while the iteration was in progress.

Next

The `Next` method returns the next element in the platform type list that created this platform type-list iterator.

```
ODPlatformType Next ();
```

return value The next element in the platform type list, or `kODNULL` if you have reached the end of the platform type list.

DISCUSSION

If your part calls this method before calling this platform type-list iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The platform type list was modified while the iteration was in progress.

SEE ALSO

The `ODPlatformType` type (page 889).

ODRecord

Superclasses `ODDescList` → `ODDesc` → `ODObject`

Subclasses `ODAppleEvent`

An object of the `ODRecord` class is a wrapper for an AE record (type `AERecord`), a descriptor list that can be used to construct Apple event parameters.

Description

An AE record is a special descriptor list that allows keyword-specified descriptor records for Apple event parameters.

For more information on Apple events and the `AERecord` type, see the "Introduction to Apple Events" chapter of *Inside Macintosh: Interapplication Communication*. For general information on scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents a summary description of the `ODRecord` method, followed by a detailed description.

`InitODRecord` Initializes this AE record.

InitODRecord

The `InitODRecord` method initializes this AE record.

```
void InitODRecord ();
```

DISCUSSION

There is no factory method for the `ODRecord` class; after creating a new AE record, OpenDoc or your part must call this method to initialize the new AE record.

ODRefCntObject

Superclasses `ODObject`

Subclasses `ODContainer`, `ODDocument`, `ODDraft`, `ODExtension`, `ODMenuBar`, `ODPersistentObject`, `ODShape`, `ODStorageUnit`, `ODTransform`, and `ODWindow`

An object of the `ODRefCntObject` class implements reference counting, a mechanism that allows OpenDoc to manage memory used by objects.

Description

In a typical OpenDoc session, various objects are shared by other objects, each of which has a reference to the shared object. A shared object should not be deleted as long as any object is using it. OpenDoc uses **reference counting** to manage memory for shared objects. Each reference-counted object keeps track of the number of existing references to it. An object's **reference count** indicates the number of objects that are currently using it. When an object's reference count drops to 0, no other object is using it; only then is it safe to delete the object, freeing the space it occupies in memory.

The `ODRefCntObject` class is the abstract superclass for all OpenDoc classes whose objects require reference counting. You should never instantiate `ODRefCntObject` itself, but you can instantiate a subclass by calling the appropriate factory method. If the factory method creates a new object, it sets the reference count for that object to 1; if the factory method returns a reference to an existing object, it increments that object's reference count by 1.

Memory management for each reference-counted class is the responsibility of its factory class. For example, the `ODStorageSystem` class is the factory class for the `ODContainer` class; that is, the storage system's factory method creates a container object. When the container object's reference count drops to 0, it informs the storage system, which deletes the container object.

Methods

This section presents summary descriptions of the `ODRefCntObject` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Managing the Reference Count

`GetRefCount`	Returns the current reference count of this object.
`Acquire`	Increments an object's reference count by 1.
`Release`	Decrements an object's reference count by 1.

Acquire

The `Acquire` method increments an object's reference count by 1.

```
void Acquire ();
```

DISCUSSION

Most methods that return a reference to a reference-counted object increment the object's reference count. However, if your part obtains a reference to a reference-counted object from a method that does not increment the object's reference count, you should call the object's `Acquire` method before you cache the reference in any data structure. When the reference is replaced or removed from the data structure, you should call the object's `Release` method to decrement its reference count.

OVERRIDING

Every subclass of `ODRefCntObject` has its own version of the `Acquire` method. Your subclass of `ODPart` overrides this method if your part performs any specific actions when its reference count is incremented. The override method must call its inherited `Acquire` method at the beginning of its implementation.

SEE ALSO

The `ODRefCntObject::Release` method (page 555).
The `ODPart` class (page 445).

GetRefCount

The `GetRefCount` method returns the current reference count of this object.

```
ODULong GetRefCount ();
```

return value The current reference count of this object.

Release

The `Release` method decrements an object's reference count by 1.

```
void Release ();
```

DISCUSSION

When you no longer need an object reference that you obtained by calling a factory method (for example, the draft's `CreatePart` or `AcquirePart` method), you should call the object's `Release` method. In addition, you should balance every call to the object's `Acquire` method with a call to its `Release` method.

OVERRIDING

Every subclass of `ODRefCntObject` has its own version of the `Release` method that tells the draft to release the object from memory if the object's reference count becomes 0. Your subclass of `ODPart` must override this method. The override method must call its inherited `Release` method at the beginning of its implementation.

EXCEPTIONS

`kODErrZeroRefCount`	The reference count cannot be decremented because the reference count is already 0.

SEE ALSO

The `ODPart` class (page 445).

ODSemanticInterface

Superclasses `ODExtension` → `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODSemanticInterface` class implements an extension that handles semantic events for your part. It is recommended, but not required, that your part editor support this extension.

Description

When a document is opened, the session object creates a single semantic interface object for the document shell. All parts of that document share the document shell's semantic interface object; you can obtain a reference to it by calling the session object's `AcquireShellSemtInterface` method (page 582).

The methods defined by the `ODSemanticInterface` class parallel handlers and functions defined for Apple events in *Inside Macintosh: Interapplication Communication*. The following information assumes a basic knowledge of handling Apple events and resolving object specifiers.

The `ODSemanticInterface` class is designed to respond to semantic events received by your part. Because OpenDoc is responsible for dispatching semantic events to your part, many of the handlers that would normally be defined by multiple functions using the Apple Event Manager are grouped conceptually in the `ODSemanticInterface` class. For example, the `CallEventHandler` method (page 568) provides a bottleneck that all Apple events sent to your part must go through.

The methods of this class consist of four types: semantic-event handlers, object accessors, object-callback functions, and other handlers. OpenDoc calls the semantic-event handlers of your subclass to handle Apple events intended for your part. The object accessors and object-callback functions are used by OpenDoc to resolve object specifiers for your part. OpenDoc calls other handlers for special purposes. Your implementation of these methods can

consist of separate handlers or a single large procedure that handles all of your part's semantic events.

The `ODSemanticInterface` class is an abstract superclass that you must subclass to create your semantic interface. OpenDoc accesses your semantic interface object by calling your part's `AcquireExtension` method (page 452), which returns a reference to the extension object. The semantic-interface extension type is identified by the constant `kODExtSemanticInterface`. If your part supports this extension, your part subclass must override the `AcquireExtension`, `HasExtension`, and `ReleaseExtension` methods and provide an appropriate implementation. For more information related to extension objects, see the `ODExtension` class description (page 208).

For more information about creating and sending Apple events to other parts, see the `ODMessageInterface` class description (page 398). For more information related to resolving object specifiers, see the `ODNameResolver` class description (page 405) and the "Resolving and Creating Object Specifier Records" chapter of *Inside Macintosh: Interapplication Communication*. For more information related to Apple event and coercion handlers, see the "Responding to Apple Events" chapter of *Inside Macintosh: Interapplication Communication*. For general information on scripting support in OpenDoc, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODSemanticInterface`.

somInit

The `somInit` method initializes the instance variables in a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODSemanticInterface`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in this semantic interface object. The SOM library calls this method when this semantic interface object is created. You must not do anything that might fail in this method. This limits you to operations like setting pointer variables to null,

setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in this semantic interface object's subclass-specific initialization method; see also the `InitSemanticInterface` method (page 576).

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODSemanticInterface`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for this semantic interface object, including any storage related to additional instance variables initialized in this semantic interface object. The SOM library calls this method when this semantic interface object is deleted; this method must not fail.

Release

The `Release` method decrements an object's reference count by 1; it is inherited from the `ODRefCntObject` class.

```
void Release ();
```

If you subclass `ODSemanticInterface`, you can override this method to release an object and reclaim valuable resources like memory. Your override method must call its inherited method at the beginning of your implementation.

The inherited `Release` method decrements this semantic interface object's reference count by 1. The inherited method may delete this semantic interface object from memory if this object's reference count becomes 0. A part editor calls this method when it no longer needs a reference to this semantic interface object.

Purge

The `Purge` method frees memory on request; it is inherited from the `ODObject` class.

```
ODSize Purge (in ODSize size);
```

Every subclass of `ODObject` can override this method and should do so if it creates caches and temporary buffers. If you subclass `ODSemanticInterface`, you must override this method or risk running out of available memory. Your override method must call its inherited method at some point in your implementation (it does not matter where). You should save the size value returned by the inherited method because you will need it to compute the value to return from your override method.

Your override of this method should free any caches, noncritical buffers, or objects (up to the amount of memory specified). Your override of this method should add the number of bytes actually freed to the number returned by the inherited method and return the sum as the total amount of memory released. OpenDoc calls this method in low-memory situations; you should not allocate memory for this operation.

Methods

This section presents summary descriptions of the `ODSemanticInterface` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Initializing

`InitSemanticInterface` Initializes this semantic interface object.

Semantic-Event Handlers

`CallEventHandler` Processes the specified Apple event object for the part.

Object Accessors

`CallObjectAccessor` Resolves the object specifier into a target and returns a reference to an OpenDoc token identifying that target.

Object-Callback Functions

`CallCountProc` Counts the number of elements of the specified type in the specified container.

`CallCompareProc` Compares two descriptors.

`CallGetMarkTokenProc`
Gets the mark token to use for marking a large series of objects.

`CallMarkProc` Marks a large series of objects using the specified mark token.

`CallAdjustMarksProc`
Unmarks a series of objects that were previously marked.

`CallDisposeTokenProc`
Deallocates any part-specific data structures stored in the specified token.

`CallGetErrDescProc` Gets a reference to the part's global error descriptor object.

Other Handlers

`CallCoercionHandler`
Coerces the specified descriptor to a different type.

`CallPredispatchProc`
Calls the predispatch handler for this semantic interface's part.

`UsingPredispatchProc`
Specifies whether the predispatch method is currently being called whenever OpenDoc receives an Apple event.

OSL Settings

`GetOSLSupportFlags`

Returns the flags that indicate which handlers this semantic interface object supports.

`SetOSLSupportFlags`

Sets the flags that indicate which handlers this semantic interface object supports.

CallAdjustMarksProc

The `CallAdjustMarksProc` method unmarks a series of objects that were previously marked.

```
void CallAdjustMarksProc (in ODPart thePart,
                          in ODSLong newStart,
                          in ODSLong newStop,
                          in ODOSLToken markToken);
```

`thePart` A reference to the part associated with this semantic interface object.

`newStart` The index that specifies the new beginning of the entries that are to remain marked.

`newStop` The index that specifies the new end of the entries that are to remain marked.

`markToken` A reference to the OpenDoc token to be used for unmarking the elements.

DISCUSSION

OpenDoc calls this method, when it is in the process of resolving an object specifier, to unmark a series of objects that were previously marked by a call to the `CallMarkProc` method. The `newStart` and `newStop` parameters indicate the beginning and end, respectively, of the range of marked objects that are to remain marked. Your override of this method should iterate over the remaining

objects and unmark them. Your override of this method should use the `markToken` parameter to identify the marked objects in the iteration.

Before OpenDoc can call this method, you must call your semantic interface object's `SetOSlSupportFlags` method and specify the flag `kAEIDoMarking` to indicate that your semantic interface supports marking. In general, your part should override the `CallGetMarkTokenProc`, `CallMarkProc`, and `CallAdjustMarksProc` methods in the following cases: it already supports the marking of objects or it expects to deal with large numbers of records that might not all fit into memory at once.

Your override of this method is responsible for deallocating any structures for the previously marked objects that were allocated by your `CallMarkProc` method. If your part's semantic interface supports the marking of objects, it should also supply a token disposal method to dispose of your mark token.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to adjust the marks as requested.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODSemanticInterface::CallDisposeTokenProc` method (page 567).
The `ODSemanticInterface::CallGetMarkTokenProc` method (page 570).
The `ODSemanticInterface::CallMarkProc` method (page 571).
The `ODSemanticInterface::SetOSLSupportFlags` method (page 577).
The `ODOSLToken` class (page 442).

CallCoercionHandler

The `CallCoercionHandler` method coerces the specified descriptor to a different type.

```
void CallCoercionHandler (in ODPart thePart,
                          in ODDesc theODDesc,
                          in ODDescType toType,
                          in ODDesc theResult);
```

`thePart`	A reference to a part associated with this semantic interface object.
`theODDesc`	A reference to a descriptor to coerce.
`toType`	The type of the coerced descriptor.
`theResult`	A reference to the resulting coerced descriptor.

DISCUSSION

OpenDoc does not chain coercion handlers together. For example, an embedded part does not inherit the coercion handler of its containing part, and a part does not inherit the coercion handler of the document shell.

EXCEPTIONS

The Apple Event Manager may throw an exception if this semantic interface does not support the specified coercion.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODDescType` type (page 895).
The `ODDesc` class (page 102).

CallCompareProc

The `CallCompareProc` method compares two descriptors.

```
void CallCompareProc (in ODPart thePart,
                      in ODDescType oper,
                      in ODOSLToken obj1,
                      in ODOSLToken obj2,
                      out ODBoolean result);
```

`thePart`	A reference to the part associated with this semantic interface object.
`oper`	The comparison operator that specifies how to compare the two objects.
`obj1`	A reference to the first object in the comparison.
`obj2`	A reference to the second object in the comparison.
`result`	A Boolean value whose meaning depends on the comparison operator.

DISCUSSION

OpenDoc calls this method during object resolution if an object specifier requires comparisons between a series of objects. Your override of this method is responsible for determining what comparisons make sense among your objects and should also be capable of comparing any two objects regardless of class type. For a list of the standard comparison operators that your method should be able to handle, see the "Resolving and Creating Object Specifier Records" chapter of *Inside Macintosh: Interapplication Communication*.

You can use the name resolver's `IsODToken` method to determine whether the `obj1` and `obj2` parameters are OpenDoc tokens or just simple descriptors.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to compare the specified objects.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODDescType` type (page 895).
The `ODNameResolver::IsODToken` method (page 411).
The `ODOSLToken` class (page 442).

CallCountProc

The `CallCountProc` method counts the number of elements of the specified type in the specified container.

```
void CallCountProc (in ODPart thePart,
                    in ODDescType desiredType,
                    in ODDescType containerClass,
                    in ODOSLToken container,
                    out ODSLong result);
```

`thePart` A reference to a part associated with this semantic interface object.

`desiredType`
The type of object to be counted.

`containerClass`
The object class of the container for the desired objects.

`container` A reference to an OpenDoc token for the container.

`result` The number of objects of the desired type in the container.

DISCUSSION

This method counts the elements in the container whose type matches the `desiredType` parameter and returns that number in the `result` parameter.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to count the specified type of object.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODDescType` type (page 895).
The `ODOSLToken` class (page 442).

CallDisposeTokenProc

The `CallDisposeTokenProc` method deallocates any part-specific data structures stored in the specified token.

```
void CallDisposeTokenProc (in ODPart thePart,
                           in ODOSLToken unneededToken);
```

`thePart` A reference to the part associated with this semantic interface object.

`unneededToken`
A reference to the OpenDoc token being deleted.

DISCUSSION

You should override this method if your token contains any part-specific data structures. Your semantic interface should also supply this method if it implements a set of marking methods. When one of your mark tokens is passed to this method, your override of this method should unmark the objects associated with that token and deallocate any necessary marking data structures.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to deallocate the specified token.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODOSLToken` class (page 442).

CallEventHandler

The `CallEventHandler` method processes the specified Apple event object for the part.

```
void CallEventHandler (in ODPart thePart,
                       in ODAppleEvent theODAppleEvent,
                       in ODAppleEvent reply);
```

`thePart` A reference to the part associated with this semantic interface object.

`theODAppleEvent` A reference to the Apple event object to be processed.

`reply` A reference to a reply Apple event object returned by the message interface object's `Send` method.

DISCUSSION

This method processes the event, resolving any object specifiers along the way, and returns an appropriate reply in the provided Apple event object. This method should be capable of handling all the Apple event objects your part supports, responding to all events your part supports, and generating an appropriate exception if the specified Apple event object is not supported.

If any of the parameters specified in the `theODAppleEvent` parameter are object specifiers, your method should resolve them using the name resolver's `Resolve` method.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to handle the specified Apple event object.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODNameResolver::Resolve` method (page 412).
The `ODMessageInterface::Send` method (page 403).
The `ODAppleEvent` class (page 41).

CallGetErrDescProc

The `CallGetErrDescProc` method gets a reference to the part's global error descriptor object.

```
void CallGetErrDescProc (in ODPart thePart,
                         out ODDesc errDesc);
```

`thePart` A reference to the part associated with this semantic interface object.

`errDesc` A reference to the part's global error descriptor object.

DISCUSSION

OpenDoc calls this method before an attempt is made to resolve an object specifier and again during object resolution if an exception arises. The first call of this method clears out the `errDesc` parameter and sets its descriptor type to `typeNull`. If an exception occurs during object resolution, the second call to this method fills in the `errDesc` parameter with the descriptor record that caused the exception.

Your override of this method should define the descriptor record as a global variable and use the same descriptor for all your exception handling. When your part receives an exception during object resolution, it can use your global descriptor record to get information about the exception.

EXCEPTIONS

The Apple Event Manager may throw an exception if this semantic interface does not support this functionality.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODDesc` class (page 102).

CallGetMarkTokenProc

The `CallGetMarkTokenProc` method gets the mark token to use for marking a large series of objects.

```
void CallGetMarkTokenProc (in ODPart thePart,
                           in ODOSLToken dContainerToken,
                           in ODDescType containerClass,
                           in ODOSLToken result);
```

`thePart` A reference to the part associated with this semantic interface object.

`dContainerToken`
A reference to the OpenDoc token that specifies the container of elements to be marked.

`containerClass`
The object class of the container for the desired objects.

`result` A reference to the OpenDoc token to be used for marking the elements.

DISCUSSION

OpenDoc calls this method to obtain the mark token to pass to your semantic interface's `CallMarkProc` and `CallAdjustMarksProc` methods. The mark token you provide should contain information to uniquely identify the series of marked objects to your part.

Before OpenDoc can call this method, you must call your semantic interface object's `SetOSlSupportFlags` method and specify the flag `kAEIDoMarking` to indicate that your semantic interface supports marking. In general, your part should override the `CallGetMarkTokenProc`, `CallMarkProc`, and `CallAdjustMarksProc` methods in the following cases: it already supports

the marking of objects or it expects to deal with large numbers of records that might not all fit into memory at once.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to return an appropriate mark token.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODDescType` type (page 895).
The `ODSemanticInterface::CallAdjustMarksProc` method (page 562).
The `ODSemanticInterface::CallMarkProc` method (page 571).
The `ODSemanticInterface::SetOSLSupportFlags` method (page 577).
The `ODOSLToken` class (page 442).

CallMarkProc

The `CallMarkProc` method marks a large series of objects using the specified mark token.

```
void CallMarkProc (in ODPart thePart,
                   in ODOSLToken dToken,
                   in ODOSLToken markToken,
                   in ODSLong index);
```

`thePart`	A reference to the part associated with this semantic interface object.
`dToken`	A reference to the OpenDoc token that identifies the object to be marked.
`markToken`	A reference to an OpenDoc token to be used for marking the elements.
`index`	The current mark count. This method should associate this index with each marked element.

DISCUSSION

OpenDoc calls this method once for each object that must be marked. Your override of this method should mark the object specified by the `dToken` parameter using the specified mark token and index. OpenDoc uses the index values to identify a range of objects to unmark when it calls the `CallAdjustMarksProc` method.

Before OpenDoc can call this method, you must call your semantic interface object's `SetOSlSupportFlags` method and specify the flag `kAEIDoMarking` to indicate that your semantic interface supports marking. In general, your part should override the `CallGetMarkTokenProc`, `CallMarkProc`, and `CallAdjustMarksProc` methods in the following cases: it already supports the marking of objects or it expects to deal with large numbers of records that might not all fit into memory at once.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to mark the specified object.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODSemanticInterface::CallAdjustMarksProc` method (page 562).
The `ODSemanticInterface::CallGetMarkTokenProc` method (page 570).
The `ODSemanticInterface::SetOSLSupportFlags` method (page 577).
The `ODOSLToken` class (page 442).

CallObjectAccessor

The `CallObjectAccessor` method resolves the object specifier into a target and returns a reference to an OpenDoc token identifying that target.

```
void CallObjectAccessor (in ODPart thePart,
                         in ODDescType desiredClass,
                         in ODOSLToken container,
```

```
                                in ODDescType containerClass,
                                in ODDescType form,
                                in ODDesc selectionData,
                                in ODOSLToken value);
```

`thePart` A reference to the part associated with this semantic interface object.

`desiredClass`
The class of the desired Apple event objects.

`container` A reference to an OpenDoc token identifying the container for the desired objects.

`containerClass`
The class of the container for the desired Apple event objects.

`form` The key form specified by the object specifier record for the object or objects to be located.

`selectionData`
A reference to a descriptor object with the key data specified by the object specifier record for the object or objects to be located.

`value` A reference to an OpenDoc token to be filled in by the object accessor being called.

DISCUSSION

OpenDoc calls this method in response to a call to the name resolver's `Resolve` method. Your override of this method should be able to resolve any of the object types supported by your part.

Use the `desiredClass` and `containerClass` parameters to identify the appropriate handler.

If your part cannot resolve an object specifier, but one of your embedded parts may be able to, your override of this method can set the `value` parameter to be equivalent to the swap token obtained from the name resolver's `CreateSwapToken` method. This is important for parts that support embedding, since your part's semantic interface must allow embedded parts to resolve objects that they know about.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to resolve the specified object.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODDescType` type (page 895).
The `ODNameResolver::CreateSwapToken` method (page 408).
The `ODNameResolver::Resolve` method (page 412).
The `ODDesc` class (page 102).
The `ODOSLToken` class (page 442).
For more information on resolving object specifiers, see the "Resolving and Creating Object Specifier Records" chapter of *Inside Macintosh: Interapplication Communication*.

CallPredispatchProc

The `CallPredispatchProc` method calls the predispatch handler for this semantic interface's part.

```
void CallPredispatchProc (in ODPart thePart,
                          in ODAppleEvent theODAppleEvent,
                          in ODAppleEvent reply);
```

`thePart` A reference to the part associated with this semantic interface object.

`theODAppleEvent` A reference to an Apple event object.

`reply` A reference to an Apple event object reply that it is appropriate for the part to return.

DISCUSSION

OpenDoc calls this method for each semantic interface registered to receive predispatched Apple event objects. This method gives your semantic interface a chance to react to events that may not be destined for your part. For example, OpenDoc does not forward recording-on and recording-off events to parts that are not direct recipients, so your part can use this method to detect those events.

Before OpenDoc calls your override of this method, you must call your semantic interface object's `UsingPredispatchProc` method and pass the value of `kODTrue` in for the `usingNotUsing` parameter. Your override of this method should not raise any exceptions under normal conditions.

EXCEPTIONS

The Apple Event Manager may throw an exception if this method is unable to receive predispatched Apple event objects.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODSemanticInterface::UsingPredispatchProc` method (page 578).
The `ODAppleEvent` class (page 41).

GetOSLSupportFlags

The `GetOSLSupportFlags` method returns the flags that indicate which handlers this semantic interface object supports.

```
ODSShort GetOSLSupportFlags ();
```

return value The flags representing the level of OSL support.

DISCUSSION

This method returns the callback flags used when a part tries to resolve an object specifier using this semantic interface object.

OVERRIDING

If you subclass `ODSemanticInterface`, you can override this method. Your override method must call its inherited method at some point in your implementation (it does not matter where).

SEE ALSO

The `ODNameResolver::Resolve` method (page 412).
The `ODSemanticInterface::SetOSLSupportFlags` method (page 577).
For more information on callback flags, see the "Resolving and Creating Object Specifier Records" chapter of *Inside Macintosh: Interapplication Communication*.

InitSemanticInterface

The `InitSemanticInterface` method initializes this semantic interface object.

```
void InitSemanticInterface (in ODPart base,
                            in ODSession session);
```

`base` A reference to a part associated with this semantic interface object.

`session` A reference to the current session object.

DISCUSSION

This method is not called directly to initialize this semantic interface object, but is called by a subclass-specific initialization method. By convention, every subclass of `ODSemanticInterface` should have a separate initialization method (for example, the `InitMySemanticInterface` method) that is called when an instance of that subclass is created. The override method may have additional parameters beyond those of the `InitSemanticInterface` method. The `InitMySemanticInterface` method should call the inherited `InitSemanticInterface` method at the beginning of its implementation.

If you subclass `ODSemanticInterface`, your subclass-specific initialization method, rather than its `somInit` method, should handle any initialization

code that can potentially fail. For example, your initialization method may attempt to allocate memory for your semantic interface.

OVERRIDING

If you subclass `ODSemanticInterface`, you must not override this method.

SetOSLSupportFlags

The `SetOSLSupportFlags` method sets the flags that indicate which handlers this semantic interface object supports.

```
void SetOSLSupportFlags (in ODSShort flags);
```

`flags` The flags representing the level of OSL support.

DISCUSSION

Your part calls this method to specify the callback flags to be used during the resolution of an object specifier.

OVERRIDING

If you subclass `ODSemanticInterface`, you can override this method. Your override method must call its inherited method at some point in your implementation (it does not matter where).

SEE ALSO

The `ODNameResolver::Resolve` method (page 412).
The `ODSemanticInterface::GetOSLSupportFlags` method (page 575).
For more information on resolving object specifiers and the use of callback flags, see the "Resolving and Creating Object Specifier Records" chapter of *Inside Macintosh: Interapplication Communication*.

UsingPredispatchProc

The `UsingPredispatchProc` method specifies whether the predispatch method is currently being called whenever OpenDoc receives an Apple event.

```
void UsingPredispatchProc (in ODBoolean usingNotUsing);
```

`usingNotUsing`
`kODTrue` if the predispatch method is currently being called whenever OpenDoc receives an Apple event, otherwise `kODFalse`.

DISCUSSION

You must call this method, passing the value of `kODTrue` to the `usingNotUsing` parameter, before OpenDoc can call your semantic interface's `CallPredispatchProc` method.

OVERRIDING

If you subclass `ODSemanticInterface`, you can override this method. Your override method must call its inherited method at some point in your implementation (it does not matter where).

SEE ALSO

The `ODSemanticInterface::CallPredispatchProc` method (page 574).

ODSession

Superclasses `ODObject`
Subclasses none

The `ODSession` class provides access to session-wide OpenDoc objects as well as the initialization and shutdown of the OpenDoc environment.

Description

When an OpenDoc document is opened, the document shell creates and initializes a single instance of `ODSession`. (A part editor should never create an instance of this class directly.) During its initialization, the session object creates a single object of several OpenDoc classes. These OpenDoc objects provide the environment for supporting and manipulating the document.

Through the session object, a part editor can obtain references to most of the OpenDoc objects: the arbitrator, the clipboard object, the dispatcher, the drag-and-drop object, the info object, the message interface, the name resolver, the name-space manager, the storage system, the translation object, the undo object, and the window-state object. The document shell can similarly obtain references to the binding object, the link manager, and the document shell's semantic interface.

The `ODSession` class also includes methods for converting between a type string (an `ODType`) and the corresponding token (an `ODTypeToken`), removing an entry from the type/token table, and accessing the name of the current user of the document. The session object also generates link update IDs, which parts use to prevent circular updating when they synchronize linked data.

For more information about type strings and tokens, see "Characters, Strings, and Tokens" (page 845). For more information about the OpenDoc objects that part editors can access, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*. For more information about linking and link update IDs, see the chapters on storage and data transfer in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODSession` class methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are called only by the document shell or container applications.

Initializing

`InitSession`	Initializes this session and creates its objects.

Accessing Global Objects

`GetArbitrator`	Returns a reference to the arbitrator for this session.
`GetBinding` [D]	Returns a reference to the binding object for this session.
`GetClipboard`	Returns a reference to the clipboard object for this session.
`GetDispatcher`	Returns a reference to the dispatcher for this session.
`GetDragAndDrop`	Returns a reference to the drag-and-drop object for this session.
`GetInfo`	Returns a reference to the info object for this session.
`GetLinkManager` [D]	Returns a reference to the link manager for this session.
`GetMessageInterface`	Returns a reference to the message interface for this session.
`GetNameResolver`	Returns a reference to the name resolver for this session.
`GetNameSpaceManager`	Returns a reference to the name-space manager for this session.
`AcquireShellSemtInterface` [D]	Returns a reference to the document shell's semantic interface object for this session.
`GetStorageSystem`	Returns a reference to the storage system for this session.
`GetTranslation`	Returns a reference to the translation object for this session.

`GetUndo`	Returns a reference to the undo object for this session.
`GetWindowState`	Returns a reference to the window-state object for this session.

Replacing Global Objects

`SetArbitrator`	Replaces the arbitrator object for this session.
`SetBinding`	Replaces the binding object for this session.
`SetClipboard`	Replaces the clipboard object for this session.
`SetDispatcher`	Replaces the dispatcher object for this session.
`SetDragAndDrop`	Replaces the drag-and-drop object for this session.
`SetInfo`	Replaces the info object for this session.
`SetLinkManager`	Replaces the link manager for this session.
`SetMessageInterface`	Replaces the message interface for this session.
`SetNameResolver`	Replaces the name resolver for this session.
`SetNameSpaceManager`	Replaces the name-space manager for this session.
`SetShellSemtInterface` [D]	Replaces the document shell's semantic interface object for this session.
`SetStorageSystem`	Replaces the storage system object for this session.
`SetTranslation`	Replaces the translation object for this session.
`SetUndo`	Replaces the undo object for this session.
`SetWindowState`	Replaces the window-state object for this session.

Converting Between Type Strings and Tokens

`Tokenize`	Converts the specified type string to a token.
`GetType`	Gets the type string corresponding to the specified token, if the token exists.
`RemoveEntry`	Removes the specified entry from the type/token table for this session.

Miscellaneous

`GetUserName`	Gets a text string identifying the current user of the document.

`UniqueUpdateID`	Returns a new update ID that is unique to this session and unlikely to be repeated on the network.

AcquireShellSemtInterface

Document shell

The `AcquireShellSemtInterface` method returns a reference to the document shell's semantic interface object for this session.

```
ODSemanticInterface AcquireShellSemtInterface ();
```

return value A reference to the document shell's semantic interface object.

DISCUSSION

This method increments the reference count of the returned semantic interface. When you have finished using that semantic interface, you should call its `Release` method.

SEE ALSO

The `ODSemanticInterface` class (page 557).
For more information on semantic interfaces, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

GetArbitrator

The `GetArbitrator` method returns a reference to the arbitrator for this session.

```
ODArbitrator GetArbitrator ();
```

return value A reference to the arbitrator.

SEE ALSO

The `ODArbitrator` class (page 43).

GetBinding

Document shell

The `GetBinding` method returns a reference to the binding object for this session.

```
ODBinding GetBinding ();
```

return value A reference to the binding object.

SEE ALSO

The `ODBinding` class (page 59).

GetClipboard

The `GetClipboard` method returns a reference to the clipboard object for this session.

```
ODClipboard GetClipboard ();
```

return value A reference to the clipboard object.

SEE ALSO

The `ODClipboard` class (page 81).

GetDispatcher

The `GetDispatcher` method returns a reference to the dispatcher for this session.

```
ODDispatcher GetDispatcher ();
```

return value A reference to the dispatcher.

SEE ALSO

The `ODDispatcher` class (page 108).

GetDragAndDrop

The `GetDragAndDrop` method returns a reference to the drag-and-drop object for this session.

```
ODDragAndDrop GetDragAndDrop ();
```

return value A reference to the drag-and-drop object.

SEE ALSO

The `ODDragAndDrop` class (page 184).

GetInfo

The `GetInfo` method returns a reference to the info object for this session.

```
ODInfo GetInfo ();
```

return value A reference to the info object.

SEE ALSO

The `ODInfo` class (page 337).

GetLinkManager

Document shell

The `GetLinkManager` method returns a reference to the link manager for this session.

```
ODLinkManager GetLinkManager ();
```

return value A reference to the link manager.

SEE ALSO

The `ODLinkManager` class (page 353).

GetMessageInterface

The `GetMessageInterface` method returns a reference to the message interface for this session.

```
ODMessageInterface GetMessageInterface ();
```

return value A reference to the message interface.

SEE ALSO

The `ODMessageInterface` class (page 398).

GetNameResolver

The `GetNameResolver` method returns a reference to the name resolver for this session.

```
ODNameResolver GetNameResolver ();
```

return value A reference to the name resolver.

SEE ALSO

The `ODNameResolver` class (page 405).

GetNameSpaceManager

The `GetNameSpaceManager` method returns a reference to the name-space manager for this session.

```
ODNameSpaceManager GetNameSpaceManager ();
```

return value A reference to the name-space manager.

SEE ALSO

The `ODNameSpaceManager` class (page 421).

GetStorageSystem

The `GetStorageSystem` method returns a reference to the storage system for this session.

```
ODStorageSystem GetStorageSystem ();
```

return value A reference to the storage system.

SEE ALSO

The `ODStorageSystem` class (page 634).

GetTranslation

The `GetTranslation` method returns a reference to the translation object for this session.

```
ODTranslation GetTranslation ();
```

return value A reference to the translation object.

SEE ALSO

The `ODTranslation` class (page 762).

GetType

The `GetType` method gets the type string corresponding to the specified token, if the token exists.

```
ODBoolean GetType (in ODTypeToken token,
                   out ODType type);
```

`token` A tokenized string representing the token of interest, expressed as a 32-bit value.

`type` The type string corresponding to the specified token.

return value `kODTrue` if the specified token exists in the type/token table for this session, otherwise `kODFalse`.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODSession::RemoveEntry` method (page 589).
The `ODSession::Tokenize` method (page 598).

GetUndo

The `GetUndo` method returns a reference to the undo object for this session.

```
ODUndo GetUndo ();
```

return value A reference to the undo object.

SEE ALSO

The `ODUndo` class (page 779).

GetUserName

The `GetUserName` method gets a text string identifying the current user of the document.

```
void GetUserName (out ODIText name);
```

`name` A text string identifying the current user of the document.

SEE ALSO

The `ODIText` type (page 845).

GetWindowState

The `GetWindowState` method returns a reference to the window-state object for this session.

```
ODWindowState GetWindowState ();
```

return value A reference to the window-state object.

SEE ALSO

The `ODWindowState` class (page 817).

InitSession

The `InitSession` method initializes this session and creates its objects.

```
void InitSession ();
```

DISCUSSION

Your part editor should never call this method directly; the document shell automatically calls this method when an OpenDoc document is opened.

RemoveEntry

The `RemoveEntry` method removes the specified entry from the type/token table for this session.

```
void RemoveEntry (in ODType type);
```

`type` The type string identifying the entry to be removed from the type/token table.

DISCUSSION

If the type string was never converted to a token, no action is taken (because the table does not contain an entry for the specified type).

SEE ALSO

The `ODSession::GetType` method (page 587).
The `ODSession::Tokenize` method (page 598).

SetArbitrator

The `SetArbitrator` method replaces the arbitrator object for this session.

```
void SetArbitrator (in ODArbitrator arbitrator);
```

`arbitrator` A reference to the new arbitrator for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODArbitrator` class (page 43).
The `ODShellPluginInstall` function (page 922).

SetBinding

The `SetBinding` method replaces the binding object for this session.

```
void SetBinding (in ODBinding binding);
```

`binding` A reference to the new binding object for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODBinding` class (page 59).
The `ODShellPluginInstall` function (page 922).

SetClipboard

The `SetClipboard` method replaces the clipboard object for this session.

```
void SetClipboard (in ODClipboard clipboard);
```

`clipboard` A reference to the new clipboard object for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODClipboard` class (page 81).
The `ODShellPluginInstall` function (page 922).

SetDispatcher

The `SetDispatcher` method replaces the dispatcher object for this session.

```
void SetDispatcher (in ODDispatcher dispatcher);
```

`dispatcher` A reference to the new dispatcher for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODDispatcher` class (page 108).
The `ODShellPluginInstall` function (page 922).

SetDragAndDrop

The `SetDragAndDrop` method replaces the drag-and-drop object for this session.

```
void SetDragAndDrop (in ODDragAndDrop dragAndDrop);
```

`dragAndDrop`
A reference to the new drag-and-drop object for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODDragAndDrop` class (page 184).
The `ODShellPluginInstall` function (page 922).

SetInfo

The `SetInfo` method replaces the info object for this session.

```
void SetInfo (in ODInfo info);
```

`info` A reference to the new info object for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODInfo` class (page 337).
The `ODShellPluginInstall` function (page 922).

SetLinkManager

The `SetLinkManager` method replaces the link manager for this session.

```
void SetLinkManager (in ODLinkManager linkManager);
```

`linkManager`
A reference to the new link manager for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODLinkManager` class (page 353).
The `ODShellPluginInstall` function (page 922).

SetMessageInterface

The `SetMessageInterface` method replaces the message interface for this session.

```
void SetMessageInterface (
                    in ODMessageInterface messageInterface);
```

`messageInterface`
A reference to the new message interface for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODMessageInterface` class (page 398).
The `ODShellPluginInstall` function (page 922).

SetNameResolver

The `SetNameResolver` method replaces the name resolver for this session.

```
void SetNameResolver (in ODNameResolver nameResolver);
```

`nameResolver`
A reference to the new name resolver for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODNameResolver` class (page 405).
The `ODShellPluginInstall` function (page 922).

SetNameSpaceManager

The `SetNameSpaceManager` method replaces the name-space manager for this session.

```
void SetNameSpaceManager (
                    in ODNameSpaceManager nameSpaceManager);
```

`nameSpaceManager`
A reference to the new name-space manager for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODNameSpaceManager` class (page 421).
The `ODShellPluginInstall` function (page 922).

SetShellSemtInterface

Document shell

The `SetShellSemtInterface` method replaces the document shell's semantic interface object for this session.

```
void SetShellSemtInterface (
             in ODSemanticInterface shellSemanticInterface);
```

`shellSemanticInterface`
A reference to the document shell's new semantic interface object.

DISCUSSION

The session releases the existing semantic interface object before acquiring a new one.

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODSemanticInterface` class (page 557).
The `ODShellPluginInstall` function (page 922).
For more information on semantic interfaces, see the chapter on semantic events and scripting in the *OpenDoc Programmer's Guide for the Mac OS*.

SetStorageSystem

The `SetStorageSystem` method replaces the storage system object for this session.

```
void SetStorageSystem (in ODStorageSystem storageSystem);
```

`storageSystem`
A reference to the new storage system object for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODStorageSystem` class (page 634).
The `ODShellPluginInstall` function (page 922).

SetTranslation

The `SetTranslation` method replaces the translation object for this session.

```
void SetTranslation (in ODTranslation translation);
```

`translation`
A reference to the new translation object for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODTranslation` class (page 762).
The `ODShellPluginInstall` function (page 922).

SetUndo

The `SetUndo` method replaces the undo object for this session.

```
void SetUndo (in ODUndo undo);
```

`undo` A reference to the new undo object for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODUndo` class (page 779).
The `ODShellPluginInstall` function (page 922).

SetWindowState

The `SetWindowState` method replaces the window-state object for this session.

```
void SetWindowState (in ODWindowState windowState);
```

`windowState`
A reference to the new window-state object for this session.

DISCUSSION

This method may be called only by a shell plug-in's `ODShellPluginInstall` function.

SEE ALSO

The `ODWindowState` class (page 817).
The `ODShellPluginInstall` function (page 922).

Tokenize

The `Tokenize` method converts the specified type string to a token.

```
ODTypeToken Tokenize (in ODType type);
```

`type`	A type string to be converted.
return value	A tokenized string representing the token corresponding to the specified type, expressed as a 32-bit value.

DISCUSSION

If the specified type string has already been converted to a token (by a previous call to the `Tokenize` method), this method returns the corresponding token from the type/token table. Otherwise, it converts the type to a token and adds a new entry to the type/token table.

Only tokens with entries in the type/token table can be converted back to type strings (by the `GetType` method).

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to generate a token.

SEE ALSO

The `ODTypeToken` type (page 847).
The `ODSession::GetType` method (page 587).
The `ODSession::RemoveEntry` method (page 589).

UniqueUpdateID

The `UniqueUpdateID` method returns a new update ID that is unique to this session and unlikely to be repeated on the network.

```
ODUpdateID UniqueUpdateID ();
```

return value	A unique update ID, expressed as a 32-bit value.

DISCUSSION

An update ID uniquely identifies a version of the clipboard content or linked content. The update ID values have no significance other than the context of testing them for equality; they remain valid only during the current session.

SEE ALSO

The `ODUpdateID` type (page 887).
For more information on linking and link update IDs, see the chapters on storage and data transfer in the *OpenDoc Programmer's Guide for the Mac OS*.

ODSettingsExtension

Superclasses `ODExtension` → `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODSettingsExtension` class represents a settings modal dialog box—an extension to the Part Info dialog box—that a part editor can create and display.

Description

The Part Info dialog box provides access only to the standard Info properties that all OpenDoc parts have. To allow the user to access properties specific to your parts, you can create a settings extension object to display a settings modal dialog box. If you implement a settings extension object, a button with the title "Settings…" appears in the lower left corner of the Part Info dialog box. When the user clicks this button, your part editor displays a modal dialog box so the user can edit part-specific settings.

The `ODSettingsExtension` class is an abstract superclass that you can subclass to create a settings extension for a modal dialog box. Once you implement it, OpenDoc accesses your settings extension object by calling your part's `AcquireExtension` method (page 452), which returns a reference to the extension object. For more information related to extension objects, see the `ODExtension` class description (page 208).

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODSettingsExtension`.

somInit

The `somInit` method initializes the instance variables in a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODSettingsExtension`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in this settings extension object. The SOM library calls this method when this settings extension object is created. You must not do anything that might fail in this method. This limits you to operations like setting pointer variables to null, setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in this settings extension object's subclass-specific initialization method; see also the `InitSettingsExtension` method (page 604).

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODSettingsExtension`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for this settings extension object, including any storage related to additional instance variables initialized in this settings extension object. The SOM library calls this method when this settings extension object is deleted; this method must not fail.

Release

The `Release` method decrements an object's reference count by 1; it is inherited from the `ODRefCntObject` class.

```
void Release ();
```

If you subclass `ODSettingsExtension`, you can override this method to release an object and reclaim valuable resources like memory. Your override

method must call its inherited method at the beginning of your implementation.

The inherited `Release` method decrements this settings extension object's reference count by 1. The inherited method may delete this settings extension object from memory (if this object's reference count becomes 0). OpenDoc calls this method when it no longer needs a reference to this settings extension object (for example, after the user has dismissed the Part Info dialog box).

Purge

The `Purge` method frees memory on request; it is inherited from the `ODObject` class.

```
ODSize Purge (in ODSize size);
```

Every subclass of `ODObject` can override this method and should do so if it creates caches and temporary buffers. If you subclass `ODSettingsExtension`, you must override this method or risk running out of available memory. Your override method must call its inherited method at some point in your implementation (it does not matter where). You should save the size value returned by the inherited method because you will need it to compute the value to return from your override method.

Your override of this method should free any caches, noncritical buffers, or objects (up to the amount of memory specified). Your override of this method should add the number of bytes actually freed to the number returned by the inherited method and return the sum as the total amount of memory released. OpenDoc calls this method in low-memory situations; you should not allocate memory for this operation.

Methods

This section presents summary descriptions of the `ODSettingsExtension` methods, followed by detailed descriptions in alphabetical order.

`InitSettingsExtension`
: Initializes this settings extension object.

`ShowSettings`	Should display a modal dialog box so the user can edit part-specific settings.

InitSettingsExtension

The `InitSettingsExtension` method initializes this settings extension object.

```
void InitSettingsExtension (in ODPart owner);
```

`owner`	A reference to this settings extension's base object.

DISCUSSION

This method is not called directly to initialize this settings extension object, but is called by a subclass-specific initialization method. By convention, every subclass of `ODSettingsExtension` should have an override method that is called when an instance of that subclass is created. The override method may have additional parameters beyond those of the inherited `InitSettingsExtension` method. The override method should call the inherited `InitSettingsExtension` method at the beginning of its implementation. The inherited `InitSettingsExtension` method in turn calls the `InitExtension` method associated with this settings extension's base object (`ODExtension`) to prepare this settings extension for use.

If you subclass `ODSettingsExtension`, your subclass-specific initialization method, rather than its `somInit` method, should handle any initialization code that can potentially fail. For example, your initialization method may attempt to allocate memory for your settings extension.

OVERRIDING

If you subclass `ODSettingsExtension`, you must override this method. Your override method must call its inherited method at the beginning of your implementation.

SEE ALSO

The `ODExtension::InitExtension` method (page 213).

ShowSettings

The `ShowSettings` method should display a modal dialog box so the user can edit part-specific settings.

```
void ShowSettings (in ODFacet facet);
```

`facet` A reference to a facet that indicates the window (and indirectly the monitor) in which to display the settings modal dialog box. The facet's frame indicates which frame of your part's properties should be displayed and edited if your part has part-specific properties that are then also frame specific.

DISCUSSION

OpenDoc calls this method.

OVERRIDING

If you subclass `ODSettingsExtension`, you must override this method. Your override method must not call its inherited method; that is, your override method must implement this method's functionality completely.

ODShape

Superclasses `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODShape` class represents a geometric shape that OpenDoc uses to manage the display of parts.

Description

Shape objects encapsulate geometric shapes. The simplest shape is an **empty** shape, which occupies no area.

Your part creates an empty shape by calling the `CreateShape` method (page 309) of a frame, the `CreateShape` method (page 236) of a facet, or the `NewShape` method (page 622) of an existing shape. Your part can create a copy of an existing shape by calling that shape's `Copy` method (page 611).

Frames and facets use the following kinds of shape objects for frame negotiation, clipping, and hit-testing.

- A **frame shape** defines the area that the containing part delegates to an embedded part's display frame.
- A **clip shape** defines the area of a facet in which drawing can occur; it is the area not obscured by overlapping content of the containing part.
- A **used shape** defines the area of an embedded part's frame that has actual content to display; the containing part is free to display its own content within the embedded frame, but it must remain outside the used shape area.
- An **active shape** defines the area of a facet within which the embedded part responds to mouse events.

For more information about frame shapes and used shapes, see the `ODFrame` class description (page 288); for information about clip shapes and active shapes, see the `ODFacet` class description (page 215). For more information on

the graphics systems available on the Mac OS platform, see *Inside Macintosh: Imaging with QuickDraw* and *Inside Macintosh: QuickDraw GX Graphics*.

Geometric Representation

The geometric representation of a shape is a description of its outline as a rectangle or polygon (or possibly as a series of curves if an advanced rendering system such as QuickDraw GX is available). A shape might not have a geometric representation if it is represented internally by a platform-specific data structure. For example, on the Mac OS platform, a shape may be represented by a QuickDraw region, which consists of a set of pixels rather than a polygonal outline.

A geometric (polygonal) representation of a shape can always be converted into a platform-specific representation. However, if a shape has no geometric representation, it cannot necessarily generate one. (On the Mac OS platform, a region-based shape can in fact be converted back to a polygon, but the conversion usually results in loss of accuracy and ugly stair-step effects on diagonal edges.)

The distinction is important because certain methods in the `ODShape` class require the shape to have a geometric (polygonal) representation. For example, for the `Transform` method (page 631) to apply a complex transformation such as a rotation, scale, or skew to a shape, the shape must have a geometric representation.

In addition, to ensure cross-platform compatibility, shapes that are stored persistently in documents and written to storage unit values (such as frame shapes) must be stored as polygons. A shape with no geometric representation cannot be stored in a document that could be moved to another platform.

Geometry Mode

The **geometry mode** of a shape object specifies whether the shape is required to maintain its geometric (polygonal) representation. The geometry mode has three possible values:

- Preserve geometry (`kODPreserveGeometry`) is the default value and indicates that the shape must maintain its polygonal representation for as long as possible. A shape's polygonal representation is lost if the shape is combined with (that is, unioned with, subtracted from, or intersected with) a

shape that does not have a polygonal representation. The shape's polygonal representation is replaced by the area that results from combining the two shapes.

- Lose geometry (`kODLoseGeometry`) indicates that the shape does not need to use a polygon to describe its geometric representation. The polygonal representation can be discarded in order to optimize speed, but at the expense of accuracy and persistent storage capability. For example, on the Mac OS platform, a shape's geometry can be represented by a QuickDraw region (rather than a polygon) to take advantage of the built-in region operators of QuickDraw, which are much faster than the polygon clipper in OpenDoc. This mode is often used for a frame's used shape, which is nonpersistent and onscreen only.
- Needs geometry (`kODNeedsGeometry`) indicates that the shape must always maintain its polygonal representation. The shape cannot be unioned with, subtracted from, or intersected with a shape that does not have polygonal representation. If you attempt to do so, the shape raises an exception and does not modify its geometric representation.

Geometric Operations

Shape objects support geometric operations such as union, intersection, and difference. A platform-specific shape-manipulation engine may be used to perform these operations. As a consequence, the results of certain operations, such as test for equality, difference, union, and intersection, may vary slightly from platform to platform; these cross-platform differences should be apparent only at the pixel level.

Methods

This section presents summary descriptions of the `ODShape` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [M] are specific to the Mac OS platform.

Creating Shapes

`NewShape`	Creates a new empty shape object.
`Copy`	Creates a new shape object that is a copy of this shape.

Manipulating Shape Geometry

`ReadShape`	Reads shape data from the specified storage unit into this shape.
`WriteShape`	Writes this shape to the specified storage unit.
`CopyFrom`	Modifies this shape to make it equivalent to the specified source shape.
`Reset`	Replaces this shape's geometric representation with an empty area.
`CopyPolygon`	Returns a copy of this shape's geometric representation, expressed as a polygon.
`SetPolygon`	Modifies this shape to make it equivalent to the specified polygon.
`GetBoundingBox`	Returns, in the specified structure, the smallest rectangle that surrounds this shape.
`SetRectangle`	Modifies this shape to make it equivalent to the specified rectangle.
`GetPlatformShape`	Returns a graphics-system-specific data structure representing this shape.
`SetPlatformShape`	Modifies this shape to make it equivalent to the specified graphics-system-specific shape.
`GetQDRegion` [M]	Returns an approximation of this shape in the form of a read-only QuickDraw region.
`CopyQDRegion` [M]	Returns an approximation of this shape in the form of a new, modifiable QuickDraw region.
`SetQDRegion` [M]	Modifies this shape to make it equivalent to the specified QuickDraw region.
`GetGXShape` [M]	Returns the geometric representation of this shape, expressed as a QuickDraw GX shape.
`SetGXShape` [M]	Modifies this shape to make it equivalent to the specified QuickDraw GX shape.

Testing the Shape

`IsEmpty`	Returns a Boolean value that indicates whether this shape is empty (occupies no area).
`IsRectangular`	Returns a Boolean value that indicates whether this shape can be described by a rectangle.

`HasGeometry`	Returns a Boolean value that specifies whether the geometric representation of this shape can be described by a polygon.
`IsSameAs`	Returns a Boolean value that indicates whether this shape is identical to the specified shape.
`ContainsPoint`	Returns a Boolean value that indicates whether the specified point is within the area of this shape.

Manipulating Geometry Mode

`GetGeometryMode`	Returns the current geometry mode of this shape.
`SetGeometryMode`	Sets the geometry mode of this shape.

Performing Geometric Operations

`Intersect`	Modifies this shape by intersecting it with the specified shape.
`Outset`	Modifies this shape by moving its boundary outwards—away from its interior—by the specified distance.
`Subtract`	Modifies this shape by subtracting the specified shape from it.
`Union`	Modifies this shape to be the union of this shape and the specified shape.
`Transform`	Modifies this shape by applying the specified transform to it.
`InverseTransform`	Modifies this shape by applying the inverse of the specified transform to it.

ContainsPoint

The `ContainsPoint` method returns a Boolean value that indicates whether the specified point is within the area of this shape.

```
ODBoolean ContainsPoint (in ODPoint point);
```

`point` The point to test, expressed in this shape's coordinate space.

return value `kODTrue` if the point is within this shape's area, otherwise `kODFalse`.

SEE ALSO

The `ODPoint` type (page 855).

Copy

The `Copy` method creates a new shape object that is a copy of this shape.

```
ODShape Copy ();
```

return value A reference to the newly created shape.

DISCUSSION

The new shape does not share any data with this shape; therefore, you can modify each of the shapes independently. This method automatically deletes the new shape if the copy operation fails.

This method initializes the reference count of the returned shape. When you have finished using that shape, you should call its `Release` method.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to copy this shape.

SEE ALSO

The `ODShape::CopyFrom` method (page 612).

CopyFrom

The `CopyFrom` method modifies this shape to make it equivalent to the specified source shape.

```
void CopyFrom (in ODShape sourceShape);
```

`sourceShape`
A reference to the source shape.

DISCUSSION

After this method executes successfully, this shape and the source shape do not share any data; therefore, you can modify each of them independently.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to update this shape.

SEE ALSO

The `ODShape::Copy` method (page 611).

CopyPolygon

The `CopyPolygon` method returns a copy of this shape's geometric representation, expressed as a polygon.

```
void CopyPolygon (out ODPolygon copy);
```

`copy`	A structure whose fields are set to represent a polygon that describes this shape's geometric representation, or an empty polygon if this shape's geometric representation cannot be represented by a polygon.

DISCUSSION

To check whether the shape's geometric representation can be described by a polygon, call the `HasGeometry` method. Note that some geometric representations, such as curves, can only be approximated by a polygon.

The polygon returned in the `copy` output parameter is not owned by this shape; you are allowed to modify it. When you no longer need the polygon, you should deallocate its storage.

EXCEPTIONS

`kODErrNoShapeGeometry`	This shape has no geometric representation, so it cannot be described as a polygon.
`kODErrOutOfMemory`	There is not enough memory to copy this shape's geometric representation.

SEE ALSO

The `ODPolygon` type (page 856).
The `ODShape::HasGeometry` method (page 618).

CopyQDRegion

Mac OS

The `CopyQDRegion` method returns an approximation of this shape in the form of a new, modifiable QuickDraw region.

```
ODRgnHandle CopyQDRegion ();
```

return value A region handle that describes this shape's geometric representation.

DISCUSSION

The returned QuickDraw region is not owned by this shape; you are allowed to modify it. When you no longer need the QuickDraw region, you are responsible for disposing of its storage.

The returned region is often only an approximation of this shape because coordinates are rounded to integers and diagonal lines become stair steps. (This shape itself is not affected.) However, QuickDraw regions are optimized for onscreen display and are required by many QuickDraw and Window Manager routines.

SEE ALSO

The `ODRgnHandle` type (page 854).

GetBoundingBox

The `GetBoundingBox` method returns, in the specified structure, the smallest rectangle that surrounds this shape.

```
void GetBoundingBox (out ODRect bounds);
```

`bounds` A rectangle describing this shape's bounding box.

SEE ALSO

The `ODRect` type (page 855).

GetGeometryMode

The `GetGeometryMode` method returns the current geometry mode of this shape.

```
ODGeometryMode GetGeometryMode ();
```

return value The geometry mode for this shape. The returned value is one of the following: `kODPreserveGeometry`, `kODLoseGeometry`, or `kODNeedsGeometry`.

DISCUSSION

The geometry mode of a shape object specifies whether the shape is required to maintain its geometric (polygonal) representation.

SEE ALSO

The `ODGeometryMode` type (page 857).
The `ODShape::SetGeometryMode` method (page 625).

GetGXShape

Mac OS

The `GetGXShape` method returns the geometric representation of this shape, expressed as a QuickDraw GX shape.

```
ODgxShape GetGXShape ();
```

return value The QuickDraw GX shape that is equivalent to this shape.

DISCUSSION

The QuickDraw GX shape returned by this method is owned by this shape; you cannot modify it. However, when you no longer need it, you are responsible for deallocating its storage.

This method calls the `GetPlatformShape` method to obtain the QuickDraw GX shape.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
The QuickDraw GX graphics system is not installed or available.

SEE ALSO

The `ODgxShape` type (page 854).
The `ODShape::GetPlatformShape` method (page 616).
The `ODShape::SetGXShape` method (page 626).

GetPlatformShape

The `GetPlatformShape` method returns a graphics-system-specific data structure representing this shape.

```
ODPlatformShape GetPlatformShape (
                    in ODGraphicsSystem graphicsSystem);
```

`graphicsSystem`
A 16-bit value specifying the graphics system you want to use for this shape. Valid graphics systems are platform dependent.

return value A 32-bit value identifying the requested graphics-system-specific shape. Before using the return value, you must cast it to a valid graphics system type (such as `RgnHandle` for QuickDraw or `gxShape` for QuickDraw GX).

DISCUSSION

The type of the returned value depends on the specified graphics system. On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`).

- If you specify the QuickDraw graphics system, the returned value is a QuickDraw region handle (type `RgnHandle`). This region handle belongs to the shape; you cannot modify it or deallocate its storage. The region and its contents are valid only until the next operation on this shape.

- If you specify the QuickDraw GX graphics system, the returned value is a QuickDraw GX shape (type `gxShape`). You cannot modify this QuickDraw GX shape. However, when you no longer need it, you are responsible for deallocating its storage.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
: This implementation of OpenDoc does not support the specified graphics system, or that graphics system is not installed or available.

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODShape::SetPlatformShape` method (page 627).

GetQDRegion

Mac OS

The `GetQDRegion` method returns an approximation of this shape in the form of a read-only QuickDraw region.

```
ODRgnHandle GetQDRegion ();
```

return value A QuickDraw region that approximates this shape.

DISCUSSION

The returned QuickDraw region is owned by this shape; you cannot modify it or deallocate its storage. The region and its content are valid only until the next operation on this shape.

This method calls the `GetPlatformShape` method to obtain the QuickDraw region.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
The QuickDraw graphics system is not installed or available.

SEE ALSO

The `ODRgnHandle` type (page 854).
The `ODShape::GetPlatformShape` method (page 616).
The `ODShape::SetQDRegion` method (page 629).

HasGeometry

The `HasGeometry` method returns a Boolean value that specifies whether the geometric representation of this shape can be described by a polygon.

```
ODBoolean HasGeometry ();
```

return value `kODTrue` if this shape's geometric representation can be described by a polygon, otherwise `kODFalse`.

DISCUSSION

Your part can use this method, before calling the `CopyPolygon` method, to verify that a shape's geometric representation can be described by a polygon.

SEE ALSO

The `ODShape::CopyPolygon` method (page 612).
The `ODShape::SetGeometryMode` method (page 625).

Intersect

The `Intersect` method modifies this shape by intersecting it with the specified shape.

```
ODShape Intersect (in ODShape sectShape);
```

`sectShape` A reference to the shape to be intersected with this shape.

return value A reference to this shape after the intersection operation.

DISCUSSION

After this method executes successfully, this shape is equivalent to the intersected area.

EXCEPTIONS

`kODErrNoShapeGeometry` The geometry mode of this shape is `kODNeedsGeometry`, but the other shape has no geometric representation.

`kODErrOutOfMemory` There is not enough memory to intersect the shapes.

InverseTransform

The `InverseTransform` method modifies this shape by applying the inverse of the specified transform to it.

```
ODShape InverseTransform (in ODTransform transform);
```

`transform` A reference to the transform whose inverse is to be applied to this shape.

return value A reference to this shape after it has been transformed.

DISCUSSION

This method is the inverse operation of the `Transform` method.

Shapes without a polygonal representation may not be transformable except by simple offsets.

EXCEPTIONS

`kODErrNoShapeGeometry`	This shape does not have enough geometric information to be transformed with the inverse of the specified transform.
`kODErrTransformErr`	The transform has no inverse.

SEE ALSO

The `ODShape::Transform` method (page 631).
The `ODTransform::Invert` method (page 748).

IsEmpty

The `IsEmpty` method returns a Boolean value that indicates whether this shape is empty (occupies no area).

```
ODBoolean IsEmpty ();
```

return value `kODTrue` if the shape is empty, otherwise `kODFalse`.

DISCUSSION

A shape that occupies no area is known as an empty shape. An empty shape is typically described by the empty rectangle (0,0,0,0). Alternatively, it can be described by a polygon with 0 contours.

IsRectangular

The `IsRectangular` method returns a Boolean value that indicates whether this shape can be described by a rectangle.

```
ODBoolean IsRectangular ();
```

return value `kODTrue` if this shape is rectangular, otherwise `kODFalse`.

DISCUSSION

Empty shapes are considered to be rectangular; they can be described by the empty rectangle (0,0,0,0).

SEE ALSO

The `ODShape::GetBoundingBox` method (page 614).
The `ODShape::SetRectangle` method (page 630).

IsSameAs

The `IsSameAs` method returns a Boolean value that indicates whether this shape is identical to the specified shape.

```
ODBoolean IsSameAs (in ODShape compareShape);
```

`compareShape`
A reference to the shape to be compared with this shape.

return value `kODTrue` if the shapes are equivalent, otherwise `kODFalse`.

DISCUSSION

If both shapes have polygonal representations, they are equivalent if they consist of the same contours. The order of the contours does not matter. Each of the corresponding contours in the two polygons must have their vertices listed in the same direction (clockwise or counterclockwise).

If this shape is described only by a platform shape (such as a region), then the platform graphics system is used to determine equality; typically, the `compareShape` parameter must be a platform shape of the same type.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to compare the two shapes.

NewShape

The `NewShape` method creates a new empty shape object.

```
ODShape NewShape ();
```

return value A reference to the newly created shape, or `kODNULL` if an error occurred.

DISCUSSION

This method initializes the reference count of the returned shape. When you have finished using that shape, you should call its `Release` method.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to create a new shape.

Outset

The `Outset` method modifies this shape by moving its boundary outwards—away from its interior—by the specified distance.

```
ODShape Outset (in ODCoordinate distance);
```

`distance` The distance (expressed in the shape's coordinate system) to move the shape's outline.

return value A reference to this shape after the outset operation.

DISCUSSION

This method is typically used to create a border around a shape. To do this, copy the original shape, outset the copy, then subtract the original shape from the copy.

To inset a shape (move the boundary inwards), call this method with a negative distance.

EXCEPTIONS

`kODErrOutOfMemory` There is not enough memory to move the shape's boundaries. The shape is not modified.

SEE ALSO

The `ODCoordinate` type (page 855).

ReadShape

The `ReadShape` method reads shape data from the specified storage unit into this shape.

```
ODShape ReadShape (in ODStorageUnit storageUnit);
```

`storageUnit`
A reference to the storage unit from which data is to be read.

return value A reference to this shape.

DISCUSSION

Before calling this method, you must focus the storage unit to the property that contains the shape data.

If the focused property contains a value of type `kODPolygon`, this method reads the shape data from that value. Otherwise, if the focused property contains a value of a platform-specific shape value type, this method reads the shape data from that value.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: This storage unit is not focused on a property or a value.

SEE ALSO

The `ODShape::WriteShape` method (page 632).

Reset

The `Reset` method replaces this shape's geometric representation with an empty area.

```
void Reset ();
```

DISCUSSION

Except for shapes created by the `Copy` method, newly created shapes start out as empty shapes; this method changes a shape back to the initial state.

The effect of this method is the same as replacing this shape with an empty rectangle by using the `SetRectangle` method with the empty rectangle (0, 0, 0, 0). However, the `Reset` method is slightly more efficient.

SEE ALSO

The `ODShape::SetRectangle` method (page 630).

SetGeometryMode

The `SetGeometryMode` method sets the geometry mode of this shape.

```
void SetGeometryMode (in ODGeometryMode mode);
```

`mode` The geometry mode for this shape. The value of `mode` must be one of the following: `kODPreserveGeometry`, `kODLoseGeometry`, or `kODNeedsGeometry`.

DISCUSSION

The geometry mode of a shape object specifies whether the shape is required to maintain its geometric (polygonal) representation.

EXCEPTIONS

`kODErrNoShapeGeometry` The specified geometry mode is `kODNeedsGeometry`, but this shape has no geometry.

SEE ALSO

The `ODGeometryMode` type (page 857).
The `ODShape::GetGeometryMode` method (page 614).

SetGXShape

Mac OS

The `SetGXShape` method modifies this shape to make it equivalent to the specified QuickDraw GX shape.

```
void SetGXShape (in ODgxShape s);
```

`s` A QuickDraw GX shape object of type rectangle, polygon, path, empty, or full.

DISCUSSION

Before calling this method, you must ensure that QuickDraw GX is installed. After this method executes successfully, the QuickDraw GX shape is owned by this shape; you cannot modify it or deallocate its storage.

This method calls the `SetPlatformShape` method to assign the shape.

EXCEPTIONS

`kODErrInvalidGraphicsSystem` The QuickDraw GX graphics system is not installed or available.

`kODErrInvalidPlatformShape` This shape's type is not one of polygon, rectangle, path, empty, or full.

`kODErrOutOfMemory` There is not enough memory to assign the QuickDraw GX shape.

SEE ALSO

The `ODgxShape` type (page 854).
The `ODShape::GetGXShape` method (page 615).
The `ODShape::SetPlatformShape` method (page 627).

SetPlatformShape

The `SetPlatformShape` method modifies this shape to make it equivalent to the specified graphics-system-specific shape.

```
void SetPlatformShape (in ODGraphicsSystem graphicsSystem,
                       in ODPlatformShape platformShape);
```

`graphicsSystem`
A 16-bit value specifying the graphics system for which you are specifying a shape. Valid graphics systems are platform dependent.

`platformShape`
A 32-bit value identifying a valid graphics-system-specific shape. Valid values for `c` are graphics-system-dependent.

DISCUSSION

Depending on the platform and graphics system you use, the specified graphics-system-specific shape may or may not be copied. On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`).

- If you specify the QuickDraw graphics system, the `platformShape` parameter is a QuickDraw region handle (type `RgnHandle`). This region handle is not copied; it is owned by this shape and you cannot modify it or deallocate its storage.
- If you specify the QuickDraw GX graphics system, the `platformShape` parameter is a QuickDraw GX shape (type `gxShape`). This QuickDraw GX shape is not copied; it is owned by this shape and you cannot modify it or deallocate its storage.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
This implementation of OpenDoc does not support the specified graphics system, or that graphics system is not installed or available.

`kODErrInvalidPlatformShape`
: Either the graphics system is QuickDraw and this shape is a null region or the graphics system is QuickDraw GX and this shape's type is not one of polygon, rectangle, path, empty, or full.

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODShape::GetPlatformShape` method (page 616).

SetPolygon

The `SetPolygon` method modifies this shape to make it equivalent to the specified polygon.

```
ODShape SetPolygon (in ODPolygon polygon);
```

`polygon` A valid polygon.

return value A reference to this shape after it has been changed to be equivalent to the specified polygon.

DISCUSSION

After this method executes successfully, you are still responsible for deleting the original polygon; this shape does not use or modify it.

If the specified polygon is self-intersecting, certain methods, such as `Intersect`, `Union`, `Subtract`, and `Outset`, will probably simplify the polygon by removing the areas of self intersection.

SEE ALSO

The `ODPolygon` type (page 856).
The `ODShape::CopyPolygon` method (page 612).

SetQDRegion

Mac OS

The `SetQDRegion` method modifies this shape to make it equivalent to the specified QuickDraw region.

```
void SetQDRegion (in ODRgnHandle rgn);
```

`rgn` A non-null QuickDraw region.

DISCUSSION

After this method executes successfully, the QuickDraw region is owned by this shape; you cannot modify it or deallocate its storage.

This method calls the `SetPlatformShape` method to assign the QuickDraw region.

EXCEPTIONS

`kODErrInvalidGraphicsSystem`
The QuickDraw graphics system is not installed or available.

`kODErrInvalidPlatformShape`
This shape is a null region.

SEE ALSO

The `ODRgnHandle` type (page 854).
The `ODShape::GetQDRegion` method (page 617).
The `ODShape::SetPlatformShape` method (page 627).

SetRectangle

The `SetRectangle` method modifies this shape to make it equivalent to the specified rectangle.

```
ODShape SetRectangle (in ODRect rect);
```

`rect`	A valid rectangle.
return value	A reference to this shape after it has been changed to be equivalent to the specified rectangle.

SEE ALSO

The `ODRect` type (page 855).

Subtract

The `Subtract` method modifies this shape by subtracting the specified shape from it.

```
ODShape Subtract (in ODShape diffShape);
```

`diffShape`	A reference to the shape to be subtracted from this shape.
return value	A reference to this shape after the subtraction operation.

DISCUSSION

After this method executes successfully, this shape is equivalent to its previous shape minus the specified shape.

EXCEPTIONS

`kODErrNoShapeGeometry`	The geometry mode of this shape is `kODNeedsGeometry`, but the other shape has no geometric representation.

`kODErrOutOfMemory`	There is not enough memory to subtract the shape.

Transform

The `Transform` method modifies this shape by applying the specified transform to it.

```
ODShape Transform (in ODTransform transform);
```

`transform`	A reference to the transform to be applied to this shape.
return value	A reference to this shape after it has been transformed.

DISCUSSION

You can use this method to move a shape from one coordinate system to another. For example, a facet's clip shape is in the coordinate system of the frame. To get it into the coordinate system of the canvas, (which you need to do in order to do QuickDraw clipping) you have to transform it by the facet's external transform.

Shapes without a geometric representation may not be transformable except by simple offsets.

EXCEPTIONS

`kODErrNoShapeGeometry`	The specified shape does not have enough geometric information to be transformed in this way.
`kODErrOutOfMemory`	There is not enough memory to transform the shape.

Union

The `Union` method modifies this shape to be the union of this shape and the specified shape.

```
ODShape Union (in ODShape unionShape);
```

`unionShape` A reference to the shape to be unioned with this shape.

return value A reference to this shape after the union operation.

DISCUSSION

After this method executes successfully, this shape is equivalent to the union of its previous shape and the specified shape.

EXCEPTIONS

`kODErrNoShapeGeometry`
: The geometry mode of this shape is `kODNeedsGeometry`, but the other shape has no geometric representation.

`kODErrOutOfMemory`
: There is not enough memory to compute the union.

WriteShape

The `WriteShape` method writes this shape to the specified storage unit.

```
void WriteShape (in ODStorageUnit storageUnit);
```

`storageUnit`
: A reference to the storage unit where the shape data is to be written.

DISCUSSION

Before calling this method, you must focus the storage unit to the property where the shape data is to be written.

If the shape can be represented as a polygon, it is written as such to the value of type `kODPolygon` in the focused property, replacing any polygon that was previously stored in that value or creating the value if it doesn't already exist.

On the Mac OS platform, a shape can always be written as a polygon, possibly after being converted from a nonpolygonal representation. On other platforms, a shape that cannot be represented as a polygon may be written in a platform-specific form to the value of the currently focused property with the appropriate platform-specific value type, replacing any shape data that was previously stored in that value or creating the value if it doesn't already exist.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to write the shape's data to the storage unit.
`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a property or a value.

SEE ALSO

The `ODShape::ReadShape` method (page 623).

ODStorageSystem

Superclasses `ODObject`

Subclasses none

An object of the `ODStorageSystem` class provides the functionality of the OpenDoc storage system.

Description

The OpenDoc storage system is a high-level persistent storage mechanism that enables multiple part editors to share a single document file effectively. When a document is opened, the session object creates a single storage-system object. All parts of that document share the storage-system object; you can obtain a reference to it by calling the session object's `GetStorageSystem` method (page 586).

The storage-system object creates and maintains a collection of container objects. Each container object can hold one or more document objects, each of which contains one or more draft objects. Each draft contains a number of storage unit objects, which hold streams of stored data.

Each container object has a container type and an identifier; these two characteristics together uniquely identify the container. OpenDoc supports both file containers and memory containers. File containers are persistent across sessions, whereas memory containers are transitory.

For more information related to container objects, see the `ODContainer` class description (page 96).

Methods

This section presents summary descriptions of the `ODStorageSystem` methods grouped according to purpose, followed by detailed descriptions in

alphabetical order. Methods marked [D] are typically called by the document shell or container applications.

Container Manipulation

`CreateContainer` [D] Creates a container object with the specified container type and identifier.

`AcquireContainer` [D] Returns a reference to the container object with the specified container type and identifier.

Storage

`CreatePlatformTypeList` Creates or copies a platform type list.

`CreateTypeList` Creates or copies a type list.

Utility Routines

`GetSession` Returns a reference to the current session object.

`NeedSpace` Notifies this storage system to reserve memory for future use.

AcquireContainer

Document shell

The `AcquireContainer` method returns a reference to the container object with the specified container type and identifier.

```
ODContainer AcquireContainer (
                        in ODContainerType containerType,
                        in ODContainerID id);
```

`containerType` The type of the container object.

`id` A container ID whose buffer contains data identifying the container object.

return value A reference to the specified container object.

DISCUSSION

The document shell and container applications call this method when opening an OpenDoc container.

The container type must be one of the following: the default file container for this platform (`kODDefaultFileContainer`), the default memory container for this platform (`kODDefaultMemoryContainer`), the Bento file container (`kODBentoFileContainer`), or the Bento memory container (`kODBentoMemoryContainer`).

The structure of the data in the `id` parameter's buffer depends on the type of container, as specified by the `containerType` parameter. For example, the identifier for a file container is a specification for a file-system file; the identifier for a memory container is a handle for a relocatable memory block.

When the structure passed as the `id` parameter is no longer needed, the caller should deallocate that structure and its buffer.

This method increments the reference count of the returned container. When the caller has finished using that container, it should call the container's `Release` method.

EXCEPTIONS

`kODErrCannotCreateContainer`
The specified container type is not valid.

SEE ALSO

The `ODContainerID` type (page 870).
The `ODContainerType` type (page 871).
The `ODStorageSystem::CreateContainer` method (page 637).

CreateContainer

Document shell

The `CreateContainer` method creates a container object with the specified container type and identifier.

```
ODContainer CreateContainer (
                        in ODContainerType containerType,
                        in ODContainerID id);
```

`containerType`
The type of the container object.

`id` A container ID whose buffer contains data identifying the container object.

return value A reference to the newly created container object.

DISCUSSION

The container type must be one of the following: the default file container for this platform (`kODDefaultFileContainer`), the default memory container for this platform (`kODDefaultMemoryContainer`), the Bento file container (`kODBentoFileContainer`), or the Bento memory container (`kODBentoMemoryContainer`).

The structure of the data in the `id` parameter's buffer depends on the type of container, as specified by the `containerType` parameter. For example, the identifier for a file container is a specification for a file-system file; the identifier for a memory container is a handle for a relocatable memory block.

The physical container corresponding to the specified container type and container identifier must exist when this method is called.

When the structure passed as the `id` parameter is no longer needed, the caller should deallocate that structure and its buffer.

This method initializes the reference count of the returned container. When the caller has finished using that container, it should call the container's `Release` method.

EXCEPTIONS

`kODErrCannotCreateContainer`
The specified container type is not valid.

`kODErrContainerExists`
A container already exists with the specified container type and container identifier.

SEE ALSO

The `ODContainerID` type (page 870).
The `ODContainerType` type (page 871).
The `ODStorageSystem::AcquireContainer` method (page 635).
The `ODContainer` class (page 96).

CreatePlatformTypeList

The `CreatePlatformTypeList` method creates or copies a platform type list.

```
ODPlatformTypeList CreatePlatformTypeList (
                            in ODPlatformTypeList typeList);
```

`typeList` A reference to the platform type list to be duplicated, or `kODNULL` to create an empty platform type list.

return value A reference to the newly created platform type list.

DISCUSSION

Your part calls this method.

SEE ALSO

The `ODPlatformTypeList` class (page 542).

CreateTypeList

The `CreateTypeList` method creates or copies a type list.

```
ODTypeList CreateTypeList (in ODTypeList typeList);
```

`typeList` A reference to the type list to be duplicated, or `kODNULL` to create an empty type list.

return value A reference to the newly created type list.

DISCUSSION

You can call this method if your part needs a list of types, for example, a list of part kinds it supports.

If the `typeList` parameter is a reference to an existing type list, the new type list is initialized to contain a copy of each element in the existing type list. The elements in the new list are in the same order as the elements of the existing type list. Because each element of the existing list is a pointer to an ISO string, the pointers themselves are not added to the new type list; instead, each string is copied and a pointer to the new copy is added to the type list.

SEE ALSO

The `ODType` type (page 846).
The `ODTypeList` class (page 770).

GetSession

The `GetSession` method returns a reference to the current session object.

```
ODSession GetSession ();
```

return value A reference to the session object that created this storage system.

DISCUSSION

Your part typically calls its storage unit's `GetSession` method instead of this method.

SEE ALSO

The `ODStorageUnit::GetSession` method (page 671).

NeedSpace

The `NeedSpace` method is called to notify this storage system to reserve memory for future use.

```
void NeedSpace (in ODSize memSize,
                in ODBoolean doPurge);
```

`memSize` The size of the memory block desired, expressed as an unsigned 32-bit value.

`doPurge` `kODTrue` if this operation should trigger the `Purge` method to obtain the requested memory, otherwise `kODFalse`.

DISCUSSION

Your part can call this method when it anticipates the need for a large memory block. This method is not guaranteed to generate the memory requested and should be used with caution as it may be a slow operation.

If memory cannot be allocated and the `doPurge` parameter is true, then this method calls the `Purge` method associated with this storage system's container objects (and transitively its document objects, draft objects, and storage-unit objects).

ODStorageUnit

Superclasses `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODStorageUnit` class represents the basic unit of persistent storage.

Description

The `ODStorageUnit` class is implemented differently for different platforms and storage mechanisms.

The set of related classes, `ODContainer` (page 96), `ODDocument` (page 130), `ODDraft` (page 145), and `ODStorageUnit` is called a **container suite**. Container suite classes are implemented as an integrated set for each platform and storage mechanism because they work intimately with one another at many levels. The container suite used by default on the Mac OS platform is the Bento container suite.

Every persistent object has an associated storage unit where it stores its data persistently. Storage units are also used for data-transfer operations; the clipboard and the drag-and-drop objects each have a content storage unit where they store the data to be transferred. Because storage units may no longer be valid as soon as a data transfer is complete, you should never cache a reference to a data-transfer object's storage unit.

Your part creates a new storage unit by calling its draft's `CreateStorageUnit` method (page 167); it accesses an existing storage unit by calling its draft's `AcquireStorageUnit` method (page 156).

A storage unit consists of one or more **properties**, each of which has a unique name within the storage unit and is distinguished by the kind of information it contains (such as, "name" or "content"). For example, the properties in your part's storage unit are used to store persistently both the content of your part and also supplemental information. OpenDoc defines a standard set of properties for all parts; you can define additional properties for information

specific to your particular part. A property consists of one or more data streams, called **values**, each of which has a named type. Each property can have only one value of a given type.

In a data-transfer operation, the source part writes data to one or more values of the data-transfer object's content storage unit; the destination part reads from those values. The source part can write either the data to be transferred, or a **promise**, which is a specification of data to be transferred at a future time. A value that contains promise data is called a promise value; a value that does not contain promise data is called a regular value. When and if a destination tries to read the promise, the storage unit first **resolves** the promise by asking the source part to fulfill it. The source part **fulfills** the promise by replacing the promise in the storage unit with the actual data that was promised.

OpenDoc assigns a runtime identifier, a **storage-unit ID**, to each of its storage units. A storage-unit ID is a nonpersistent identifier for a storage unit that is unique within its draft (storage-unit IDs are not unique across drafts and do not persist across sessions). You can use the storage-unit ID to identify storage units and to compare two storage units for equality. The ID of the storage unit for a persistent object is also used as the ID of the object itself. If you retain the ID of a persistent object when you release it, you can use that ID to re-create the object from the data stored in its storage unit. For example, when a frame is scrolled out of view, you can save its ID and release it; if the frame is later scrolled back into view, you can obtain a reference to it by passing the saved ID to the draft's `AcquireFrame` method (page 151). If the frame has been purged since you released it, the `AcquireFrame` method re-creates it from its stored data.

Storage units can maintain persistent references to other storage units in the same draft. A **persistent reference** stored in a storage unit value is an opaque type that identifies another storage unit in the same draft. Whereas the ID of a storage unit identifies that storage unit within the current session, a persistent reference to the storage unit identifies it persistently across sessions. Persistent references permit complex runtime relationships between objects to be stored externally and later reconstructed; for example, the embedding relationships of the parts within a draft are preserved by persistent references in the parts' storage units. Persistent references can be either strong or weak. In a clone operation, copies are made of all storage units referenced by strong persistent references in the object being cloned. A weak persistent reference is typically ignored during cloning; however, if a storage unit is cloned because there are strong persistent references to it, then weak persistent references to the storage unit are preserved.

OpenDoc allows you to **focus** a storage unit on the particular data of interest, called the **focus context**. Before reading or writing to a storage unit, for example, you must focus on the data stream defined by a particular value of a particular property. At any time the storage unit can be in one of the following states:

- Unfocused. When a storage unit is unfocused, its focus context is undefined.
- Focused on the entire storage unit. When the focus context is the storage unit, the data of interest includes all properties and all their values.
- Focused on a particular property. When the focus context is a property, the data of interest includes all values of the focused property.
- Focused on a particular value of a particular property. When the focus context is a value, the data of interest is the data stream corresponding to a focused value.

To define a focus context, you can specify a property by its name or a position code. Within a given property, you can specify the value of interest by a value type, a position code, or a value index. A position code is a constant that specifies either an absolute position for the property or value or a position relative to the property or value in the current focus context. Position codes allow you to access the next or previous property within the set of properties in a storage unit or the next or previous value within the set of values in the same property. A value index is a number representing the ordinal position of the value within the property. The first value created for a property has index 1; the second, 2; and so on.

When the storage unit is focused on a value of a property, you can read data from and write data to the corresponding data stream. The storage unit has a zero-based offset that specifies the current read/write position in the stream. When the storage unit is first focused, the offset is 0, indicating the beginning of the stream. Each read and write advances the offset by the number of bytes that were read or written. The storage unit also has methods that allow you to get and set the current offset.

You can call methods of a storage unit to create related objects that make it easy to work with the data in the storage unit.

- A storage-unit view represents the storage unit prefocused on a particular focus context. You can pass a storage-unit view among your software components to give them access to the particular focused data stream.

- A storage-unit cursor represents a focus context. You can create storage-unit cursors for focus contexts that you access frequently, then use those cursors to switch the focus from one context to another.
- A storage-unit reference iterator allows you to access all persistent references in the currently focused value of the storage unit.

For more information about these objects, see the descriptions of the classes `ODStorageUnitView` (page 700), `ODStorageUnitCursor` (page 691), and `ODStorageUnitRefIterator` (page 696).

Methods

This section presents summary descriptions of the `ODStorageUnit` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Access and Storage

`Lock`	Locks this storage unit for exclusive access.
`Unlock`	Unlocks this storage unit.
`CloneInto`	Copies all properties and values of this storage unit to the specified destination storage unit.
`Externalize`	Resolves all promises in this storage unit and saves all its properties and values to persistent, external storage.
`Internalize`	Reads all properties and values from this storage unit into memory.

Storage Unit Manipulation

`AddProperty`	Adds a property with the specified name to this storage unit.
`AddValue`	Adds a value of the specified type to the currently focused property.
`Remove`	Removes all properties and values in the current focus context from this storage unit.
`CountProperties`	Returns the number of properties in this storage unit.
`CountValues`	Returns the number of values in the current focus context for this storage unit.

`GetID`	Returns the storage-unit ID for this storage unit.
`GetName`	Returns the name of this storage unit.
`SetName`	Sets the name of this storage unit.
`GetSize`	Returns the size (number of bytes) of the data in the current focus context.

Focus Manipulation

`Exists`	Returns a Boolean value that indicates whether the specified focus context exists in this storage unit.
`ExistsWithCursor`	Returns a Boolean value that indicates whether the focus context represented by the specified storage-unit cursor exists in this storage unit.
`Focus`	Focuses this storage unit on the specified focus context.
`FocusWithCursor`	Focuses this storage unit on the focus context represented by the specified storage-unit cursor.
`GetProperty`	Returns the name of the property in the current focus context.

Value Manipulation

`GetValue`	Reads data from the currently focused value, starting at the offset (inclusive).
`SetValue`	Writes data to the currently focused value, starting at the offset (inclusive).
`DeleteValue`	Deletes data from the currently focused value, starting at the offset (inclusive).
`InsertValue`	Inserts data into the currently focused value, starting at the offset (inclusive).
`GetOffset`	Returns the offset of the currently focused value.
`SetOffset`	Sets the offset of the currently focused value.
`GetType`	Returns the type of the currently focused value.
`SetType`	Sets the type of the currently focused value.
`GetGenerationNumber`	Returns the generation number of the currently focused value.

`IncrementGenerationNumber`
: Increments and returns the generation number of the currently focused value.

Persistent Reference Manipulation

`GetIDFromStorageUnitRef`
: Returns the storage-unit ID of a referenced storage unit.

`GetStrongStorageUnitRef`
: Creates a strong persistent reference to the specified storage unit.

`GetWeakStorageUnitRef`
: Creates a weak persistent reference to the specified storage unit.

`IsStrongStorageUnitRef`
: Returns a Boolean value that indicates whether the specified persistent reference is a strong persistent reference.

`IsWeakStorageUnitRef`
: Returns a Boolean value that indicates whether the specified persistent reference is a weak persistent reference.

`IsValidStorageUnitRef`
: Returns a Boolean value that indicates whether the specified persistent reference is valid.

`SetStorageUnitRef`
: Creates in the currently focused value a persistent reference to the specified storage unit using the specified the persistent identifier.

`RemoveStorageUnitRef`
: Makes a persistent reference invalid in the currently focused value.

Promise Manipulation

`IsPromiseValue`
: Returns a Boolean value that indicates whether the currently focused value is a promise value.

`GetPromiseValue`
: Reads promise data from the specified value of the currently focused property.

`SetPromiseValue`	Writes data to the specified value of the currently focused property, creating the value if it doesn't exist and making the value a promise value.
`ClearAllPromises`	Changes all promise values in this storage unit into regular values.
`ResolveAllPromises`	Resolves all promises in this storage unit.

Creating Objects

`CreateCursor`	Creates a storage-unit cursor representing the specified focus context for this storage unit.
`CreateCursorWithFocus`	Creates a storage-unit cursor representing the current focus context of this storage unit.
`CreateStorageUnitRefIterator`	Creates a storage-unit reference iterator for the currently focused value.
`CreateView`	Creates a storage-unit view for this storage unit with its current focus context.

Utility Routines

`GetDraft`	Returns a reference to the draft object that created this storage unit.
`GetSession`	Returns a reference to the session object in which this storage unit runs.

AddProperty

The `AddProperty` method adds a property with the specified name to this storage unit.

```
ODStorageUnit AddProperty (in ODPropertyName propertyName);
```

`propertyName`
The name of the property to be added.

return value A reference to this storage unit.

DISCUSSION

If the storage unit does not already contain a property with the specified name, the new property is added and this storage unit is focused on the newly added property. Otherwise, this storage unit is focused on the existing property with the specified name.

EXCEPTIONS

`kODErrCannotAddProperty`
: Cannot add the specified property to this storage unit.

`kODErrIllegalNullPropertyInput`
: The specified property name is null.

`kODErrZeroRefCount`
: This storage unit has a reference count of 0.

SEE ALSO

The `ODPropertyName` type (page 874).
The `ODStorageUnit::GetProperty` method (page 670).
The `ODStorageUnit::Remove` method (page 682).

AddValue

The `AddValue` method adds a value of the specified type to the currently focused property.

```
ODStorageUnit AddValue (in ODValueType type);
```

`type`
: The type of value to be added.

return value
: A reference to this storage unit.

DISCUSSION

If the focused property does not already contain a value of the specified type, the new value is added and this storage unit is focused on the newly added

value. Otherwise, this storage unit is focused on the existing value with the specified value type.

EXCEPTIONS

`kODErrIllegalNullValueTypeInput`
: The specified value type is null.

`kODErrInvalidValueType`
: The specified value type is improperly formed or illegal.

`kODErrUnfocusedStorageUnit`
: This storage unit is not focused on a property.

`kODErrZeroRefCount`
: This storage unit has a reference count of 0.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageUnit::Remove` method (page 682).

ClearAllPromises

The `ClearAllPromises` method changes all promise values in this storage unit into regular values.

```
void ClearAllPromises ();
```

DISCUSSION

This method does not change the data in any of the promise values; it simply changes the values from promise values to regular values.

After this method executes successfully, the storage unit is unfocused.

EXCEPTIONS

`kODErrZeroRefCount`
: This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::ResolveAllPromises` method (page 684).

CloneInto

The `CloneInto` method copies all properties and values of this storage unit to the specified destination storage unit.

```
void CloneInto (in ODDraftKey key,
                in ODStorageUnit destStorageUnit,
                in ODID scopeID);
```

`key` The draft key identifying this cloning operation.

`destStorageUnit`
A reference to the destination storage unit to which the data is to be copied.

`scopeID` The ID of the frame that defines the scope of this cloning operation.

DISCUSSION

This method is not called by parts. Your part should call its draft's `Clone` or `WeakClone` method instead of this method.

If this storage unit has persistent references to other objects, the `scopeID` parameter determines which of the referenced objects are within the scope of this cloning operation. Typically, the `scopeID` parameter is the ID of a frame, and only those objects embedded in that frame are within scope. In the rare case in which the `scopeID` parameter is `kODIDAll`, all referenced objects are within scope.

This method copies this storage unit's data into the specified destination storage unit. If this storage unit has persistent references to other objects, this method clones any persistently referenced objects that are within the scope of this cloning operation. Objects referenced by strong persistent references are strongly cloned by recursive calls to the `Clone` method; objects referenced by

weak persistent references are weakly cloned by calls to the `WeakClone` method.

EXCEPTIONS

`kODErrInvalidDraftKey`	The specified draft key is not the draft key for the current cloning transaction.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODDraftKey` type (page 872).
The `ODID` type (page 869).
The `ODDraft::Clone` method (page 159).
The `ODDraft::WeakClone` method (page 181).

CountProperties

The `CountProperties` method returns the number of properties in this storage unit.

```
ODULong CountProperties ();
```

return value The number of properties in this storage unit, expressed as an unsigned 32-bit value.

DISCUSSION

After this method executes successfully, the focus context of this storage unit is not changed.

EXCEPTIONS

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::CountValues` method (page 652).

CountValues

The `CountValues` method returns the number of values in the current focus context for this storage unit.

```
ODULong CountValues ();
```

return value The number of values in this storage unit, expressed as an unsigned 32-bit value.

DISCUSSION

This storage unit must be focused on a property or a value, not the entire storage unit.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a property or a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::CountProperties` method (page 651).

CreateCursor

The `CreateCursor` method creates a storage-unit cursor representing the specified focus context for this storage unit.

```
ODStorageUnitCursor CreateCursor (
                            in ODPropertyName propertyName,
                            in ODValueType valueType,
                            in ODValueIndex valueIndex);
```

`propertyName`
: The name of the property in the focus context, or `kODNULL` if the focus context is the entire storage unit.

`valueType`
: The value type of the value in the focus context, or `kODNULL` to ignore this parameter.

`valueIndex`
: The value index of the value in the focus context, or 0 to ignore this parameter.

return value
: A reference to the newly created storage-unit cursor.

DISCUSSION

Your part calls this method; its parameters specify the focus context for the storage-unit cursor.

- To specify the entire storage unit as the focus context, pass `kODNULL` as the property name, `kODTypeAll` as the value type parameter, and `kODIndexAll` as the value index parameter.
- To specify a property as the focus context, pass its name as the property name parameter, `kODTypeAll` as the value type parameter, and `kODIndexAll` as the value index parameter.
- To specify a value as the focus context, pass the name of the property containing the value as the property name parameter. You can specify the value by either its type or its index:
  - To use type, pass the type of the desired value as the value type parameter and 0 as the value index parameter.
  - To use index, pass `kODNULL` as the value type parameter and the index of the desired value as the value index parameter.

EXCEPTIONS

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODPropertyName` type (page 874).
The `ODValueIndex` type (page 874).
The `ODValueType` type (page 874).
The `ODStorageUnit::CreateCursorWithFocus` method (page 654).
The `ODStorageUnitCursor` class (page 691).

CreateCursorWithFocus

The `CreateCursorWithFocus` method creates a storage-unit cursor representing the current focus context of this storage unit.

```
ODStorageUnitCursor CreateCursorWithFocus ();
```

return value A reference to the newly created storage-unit cursor.

EXCEPTIONS

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::CreateCursor` method (page 653).
The `ODStorageUnitCursor` class (page 691).

CreateStorageUnitRefIterator

The `CreateStorageUnitRefIterator` method creates a storage-unit reference iterator for the currently focused value.

```
ODStorageUnitRefIterator CreateStorageUnitRefIterator ();
```

return value A reference to a storage-unit reference iterator used to traverse all persistent references in the currently focused value.

DISCUSSION

You can call this method if you need to perform an operation on all storage-unit references in the currently focused value.

While you are using the returned storage-unit reference iterator, you must not modify the focused value; in particular, you must not call any of the following methods of this storage unit: `DeleteValue`, `Internalize`, `Remove`, `RemoveStorageUnitRef`, `SetStorageUnitRef`, `SetType`, or `SetValue`. Furthermore, you must not delete this storage unit.

You must delete the returned storage-unit reference iterator when it is no longer needed.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitRefIterator` class (page 696).

CreateView

The `CreateView` method creates a storage-unit view for this storage unit with its current focus context.

```
ODStorageUnitView CreateView ();
```

return value A reference to the newly created storage-unit view.

DISCUSSION

Your part calls this method.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a property or a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitView` class (page 700).

DeleteValue

The `DeleteValue` method deletes data from the currently focused value, starting at the offset (inclusive).

```
void DeleteValue (in ODULong length);
```

`length` The number of bytes of data to delete, expressed as an unsigned 32-bit value.

DISCUSSION

You call this method to delete data from the currently focused value. If that value is a promise value, the promise is fulfilled before the data is deleted. This method starts deleting data at the current offset and stops after deleting the number of bytes specified by the `length` parameter or after reaching the end of the data in the currently focused value, whichever comes first.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::InsertValue` method (page 677).
The `ODStorageUnit::Remove` method (page 682).

Exists

The `Exists` method returns a Boolean value that indicates whether the specified focus context exists in this storage unit.

```
ODBoolean Exists (in ODPropertyName propertyName,
                  in ODValueType valueType,
                  in ODValueIndex valueIndex);
```

`propertyName` The name of the property in the focus context, or `kODNULL` for the currently focused property.

`valueType` The value type of the value in the focus context, or `kODNULL` to ignore this parameter.

`valueIndex` The value index of the value in the focus context, or 0 to ignore this parameter.

return value `kODTrue` if the specified focus context exists in this storage unit, otherwise `kODFalse`.

DISCUSSION

You can call this method to see whether you can focus this storage unit on the specified focus context; however, this method does not change the current focus context. The parameters specify the focus context to be checked.

- To specify a property as the focus context, pass its name as the property name parameter or `kODNULL` for the currently focused property. Pass `kODTypeAll` as the value type parameter and `kODIndexAll` as the value index parameter.
- To specify a value as the focus context, pass the name of the property containing the value as the property name parameter or `kODNULL` for the currently focused property. You can specify the value by either its type or its index:
  - To use type, pass the type of the desired value as the value type parameter and 0 as the value index parameter.
  - To use index, pass `kODNULL` as the value type parameter and the index of the desired value as the value index parameter.

If this method returns true, it is safe to call the `Focus` method with the specified focus context.

EXCEPTIONS

`kODErrIllegalPropertyName` The specified property name is improperly formed or illegal.

`kODErrInvalidValueType` The specified value type is improperly formed or illegal.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODPropertyName` type (page 874).
The `ODValueIndex` type (page 874).

The `ODValueType` type (page 874).
The `ODStorageUnit::ExistsWithCursor` method (page 659).
The `ODStorageUnit::Focus` method (page 660).

ExistsWithCursor

The `ExistsWithCursor` method returns a Boolean value that indicates whether the focus context represented by the specified storage-unit cursor exists in this storage unit.

```
ODBoolean ExistsWithCursor (in ODStorageUnitCursor cursor);
```

`cursor`	A reference to the storage-unit cursor representing the focus context to be tested.
return value	`kODTrue` if the focus context specified by the storage-unit cursor exists in this storage unit, otherwise `kODFalse`.

DISCUSSION

You can call this method to see whether you can focus this storage unit using the specified storage-unit cursor; however, this method does not change the current focus context.

If this method returns true, it is safe to call the `FocusWithCursor` method with the specified storage-unit cursor.

EXCEPTIONS

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::Exists` method (page 657).
The `ODStorageUnit::FocusWithCursor` method (page 663).

Externalize

The `Externalize` method resolves all promises in this storage unit and saves all properties and values to persistent, external storage.

```
ODStorageUnit Externalize ();
```

return value A reference to this storage unit.

EXCEPTIONS

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::Internalize` method (page 678).

Focus

The `Focus` method focuses this storage unit on the specified focus context.

```
ODStorageUnit Focus (in ODPropertyName propertyName,
                     in ODPositionCode propertyPosCode,
                     in ODValueType valueType,
                     in ODValueIndex valueIndex,
                     in ODPositionCode valuePosCode);
```

`propertyName`
The name of the property in the desired focus context, or `kODNULL` to ignore this parameter.

`propertyPosCode`
The position code, relative to the current focus context, of the property in the desired focus context, or `kODPosUndefined` to ignore this parameter.

`valueType` The value type of the value in the desired focus context, or `kODNULL` to ignore this parameter.

`valueIndex` The value index of the value in the desired focus context, or 0 to ignore this parameter.

`valuePosCode` The position code, relative to the current focus context, of the value in the desired focus context, or `kODPosUndefined` to ignore this parameter.

return value A reference to this storage unit, focused on the specified focus context.

DISCUSSION

Your part calls this method; its parameters specify the desired focus context.

- To focus on the entire storage unit, pass `kODNULL` as the property name parameter, `kODPosAll` as the property position code parameter, `kODTypeAll` as the value type parameter, 0 as the value index parameter, and `kODPosUndefined` as the value position code parameter.
- To focus on a property, either pass its name as the property name parameter or pass `kODNULL` as the property name parameter and the appropriate code as the property position code parameter. Pass `kODTypeAll` as the value type parameter, 0 as the value index parameter, and `kODPosUndefined` as the value position code parameter.
- To focus on a value, specify the property containing the value using either the property name parameter or the property position code parameter, as described in the previous item. You can specify the value by its type, its index, or its position.
  - To use type, pass the type of the desired value as the value type parameter, 0 as the value index parameter, and `kODPosUndefined` as the value position code parameter.
  - To use index, pass `kODNULL` as the value type parameter, the index of the desired value as the value index parameter, and `kODPosUndefined` as the value position code parameter.
  - To use the position code, pass `kODNULL` as the value type parameter, 0 as the value index parameter, and the appropriate code as the value position code parameter.

After this method executes successfully, this storage unit's offset is 0.

Before calling this method, you can call the `Exists` method to check whether the specified focus context exists.

EXCEPTIONS

`kODErrIllegalPropertyName`
: The specified property name is improperly formed or illegal.

`kODErrPropertyDoesNotExist`
: This storage unit does not contain the specified property.

`kODErrSUValueDoesNotExist`
: This storage unit does not contain the specified value.

`kODErrUnsupportedPosCode`
: One of the specified position codes is not supported.

`kODErrValueIndexOutOfRange`
: The specified property has no value at the specified index.

`kODErrZeroRefCount`
: This storage unit has a reference count of 0.

SEE ALSO

The `ODPositionCode` type (page 885).
The `ODPropertyName` type (page 874).
The `ODValueIndex` type (page 874).
The `ODValueType` type (page 874).
The `ODStorageUnit::Exists` method (page 657).
The `ODStorageUnit::FocusWithCursor` method (page 663).

FocusWithCursor

The `FocusWithCursor` method focuses this storage unit on the focus context represented by the specified storage-unit cursor.

```
ODStorageUnit FocusWithCursor (
                              in ODStorageUnitCursor cursor);
```

`cursor` A reference to the storage-unit cursor representing the desired focus context.

return value A reference to this storage unit, focused on the specified focus context.

DISCUSSION

After this method executes successfully, this storage unit's offset is 0.

Before calling this method, you can call the `ExistsWithCursor` method to check whether the specified focus context exists.

EXCEPTIONS

`kODErrIllegalNullSUCursorInput`
The `cursor` parameter is null.

`kODErrPropertyDoesNotExist`
The `cursor` parameter specifies a property that does not exist.

`kODErrSUValueDoesNotExist`
The `cursor` parameter specifies a value type that does not exist for the specified property.

`kODErrValueIndexOutOfRange`
The `cursor` parameter specifies a value index that is out of the range for the specified property.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::ExistsWithCursor` method (page 659).
The `ODStorageUnit::Focus` method (page 660).

GetDraft

The `GetDraft` method returns a reference to the draft object that created this storage unit.

```
ODDraft GetDraft ();
```

return value A reference to the draft object that created this storage unit.

DISCUSSION

This method does not increment the reference count of the returned draft. For that reason, if you cache the returned draft, you should call its `Acquire` method to increment its reference count and then call its `Release` method when you are finished using it.

EXCEPTIONS

`kODErrZeroRefCount` This storage unit has a reference count of 0.

GetGenerationNumber

The `GetGenerationNumber` method returns the generation number of the currently focused value.

```
ODULong GetGenerationNumber ();
```

return value The generation number of the currently focused value, expressed as an unsigned 32-bit value.

DISCUSSION

You can use the generation number of a value to tell whether the data in the value has changed. For example, your part could compare the number returned by this method with a saved generation number.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::IncrementGenerationNumber` method (page 676).

GetID

The `GetID` method returns the storage-unit ID for this storage unit.

```
ODID GetID ();
```

return value The storage-unit ID for this storage unit.

EXCEPTIONS

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODID` type (page 869).
The `ODStorageUnit::GetIDFromStorageUnitRef` method (page 666).

GetIDFromStorageUnitRef

The `GetIDFromStorageUnitRef` method returns the storage-unit ID of a referenced storage unit.

```
ODStorageUnitID GetIDFromStorageUnitRef (
                                    in ODStorageUnitRef aRef);
```

`aRef`	A persistent storage-unit reference.
return value	The storage-unit ID of the storage unit referenced by the `aRef` parameter.

DISCUSSION

Before you call this method, you must focus this storage unit on the value that created the specified persistent reference. This method looks up the storage unit referenced by the specified persistent reference and returns its storage-unit ID.

EXCEPTIONS

`kODErrInvalidStorageUnitRef`	The specified persistent reference is not valid.
`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODStorageUnitRef` type (page 873).
The `ODStorageUnit::GetID` method (page 665).

GetName

The `GetName` method returns the name of this storage unit.

```
ODStorageUnitName GetName ();
```

return value The name of this storage unit, or `kODNULL` if the storage unit does not have a name.

DISCUSSION

When you no longer need the returned name, you should deallocate it.

EXCEPTIONS

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitName` type (page 873).
The `ODStorageUnit::SetName` method (page 685).

GetOffset

The `GetOffset` method returns the offset of the currently focused value.

```
ODULong GetOffset ();
```

return value The offset (in bytes) of the read/write position from the beginning of the data stream in the currently focused value, expressed as an unsigned 32-bit value.

DISCUSSION

An offset of 0 means the beginning of the data stream corresponding to the focused value; an offset equal to the size returned by the `GetSize` method means the end of the data stream.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::GetSize` method (page 671).
The `ODStorageUnit::SetOffset` method (page 685).

GetPromiseValue

The `GetPromiseValue` method reads promise data from the specified value of the currently focused property.

```
ODULong GetPromiseValue (in ODValueType valueType,
                         in ODULong offset,
                         in ODULong length,
                         out ODByteArray value,
                         out ODPart sourcePart);
```

`valueType`	The type of the value from which the promise data is to be read.
`offset`	The offset from which the promise data is to be retrieved, expressed as an unsigned 32-bit number of bytes from the beginning of the value.
`length`	The length (in bytes) of the data to be retrieved, expressed as an unsigned 32-bit value.
`value`	A byte array structure to contain the retrieved promise data.

`sourcePart` A reference to the part that made the promise.

return value The number of bytes read, expressed as an unsigned 32-bit value.

DISCUSSION

You call this method to read promise data without fulfilling the promise. This method first focuses the storage unit on the specified value of the currently focused property. It then starts reading data at the specified offset and stops after reading the number of bytes specified by the `length` parameter or after reaching the end of the data in the focused value, whichever comes first.

When you call this method, the `_buffer` field of the `value` output parameter should be `kODNULL`; if it isn't, the buffer to which that field points will not be deallocated.

This method sets the `_buffer` field of the `value` output parameter to point to a memory block containing the promise data, the `_maximum` field to the specified length, and the `_length` field to the number of bytes actually read.

This method sets the `sourcePart` output parameter to a reference to the part that made the promise. This method does not increment the reference count of the part that made the promise.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrInvalidValueType`
The specified value type is improperly formed or illegal.

`kODErrSUValueDoesNotExist`
This currently focused property does not have a value with the specified value type.

`kODErrUnfocusedStorageUnit`
This storage unit is not focused on a property or a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODValueType` type (page 874).
The `ODStorageUnit::IsPromiseValue` method (page 679).
The `ODStorageUnit::SetPromiseValue` method (page 686).

GetProperty

The `GetProperty` method returns the name of the property in the current focus context.

```
ODPropertyName GetProperty ();
```

return value The name of the property in the current focus context.

DISCUSSION

When you no longer need the returned property name, you should deallocate it.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a property or a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODPropertyName` type (page 874).
The `ODStorageUnit::AddProperty` method (page 647).
The `ODStorageUnit::Remove` method (page 682).

GetSession

The `GetSession` method returns a reference to the session object in which this storage unit runs.

```
ODSession GetSession ();
```

return value A reference to the session object in which this storage unit runs.

EXCEPTIONS

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageSystem::GetSession` method (page 671).

GetSize

The `GetSize` method returns the size (number of bytes) of the data in the current focus context.

```
ODULong GetSize ();
```

return value The size (in bytes) of the data in the current focus context, expressed as an unsigned 32-bit value.

DISCUSSION

If this storage unit is focused on the entire storage unit, this method returns the total size of all properties and values in this storage unit. If it is focused on a property, this method returns the total size of all values in the focused property. If it is focused on a value, this method returns the size of the data stream corresponding to the focused value.

If the currently focused value is a promise value, the promise is fulfilled before the size of the value is evaluated.

EXCEPTIONS

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

GetStrongStorageUnitRef

The `GetStrongStorageUnitRef` method creates a strong persistent reference to the specified storage unit.

```
void GetStrongStorageUnitRef (
                              in ODStorageUnitID embeddedSUID,
                              out ODStorageUnitRef strongRef);
```

`embeddedSUID`
The storage-unit ID of the storage unit whose persistent reference is desired.

`strongRef` A persistent reference to the storage unit specified by the `embeddedSUID` parameter.

DISCUSSION

Before you call this method, you should focus this storage unit on the value where you want to store the strong persistent reference. After this method executes successfully, call the `SetValue` method to store the resulting persistent reference, returned in the `strongRef` output parameter, into the currently focused value.

IMPORTANT

The scope of a persistent reference is limited to the value in which it was created. If you store the persistent reference in a different value, it will almost certainly not refer to the correct storage unit. ▲

EXCEPTIONS

`kODErrIllegalNullStorageUnitInput`	The `embeddedSUID` parameter is null.

`kODErrUnfocusedStorageUnit`
This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODStorageUnitRef` type (page 873).
The `ODStorageUnit::GetWeakStorageUnitRef` method (page 675).
The `ODStorageUnit::IsStrongStorageUnitRef` method (page 679).
The `ODStorageUnit::SetValue` method (page 689).
For more information on persistent references, see the chapter on storage in the *OpenDoc Programmer's Guide for the Mac OS*.

GetType

The `GetType` method returns the type of the currently focused value.

```
ODValueType GetType ();
```

return value The type of the currently focused value.

DISCUSSION

When you no longer need the returned value type, you should deallocate it.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageUnit::SetType` method (page 688).

GetValue

The `GetValue` method reads data from the currently focused value, starting at the offset (inclusive).

```
ODULong GetValue (in ODULong length,
                  out ODByteArray value);
```

`length`	The length (number of bytes) of data to read, expressed as an unsigned 32-bit value.
`value`	A byte array structure to contain the retrieved data.
return value	The number of bytes read, expressed as an unsigned 32-bit value.

DISCUSSION

You call this method to read data from the currently focused value. If that value is a promise value, the promise is fulfilled before the data is read. This method starts reading data at the current offset and stops after reading the number of bytes specified by the `length` parameter or after reaching the end of the data in the currently focused value, whichever comes first.

When you call this method, the `_buffer` field of the `value` output parameter should be `kODNULL`; if it isn't, the buffer to which that field points will not be deallocated.

This method sets the `_buffer` field of the `value` output parameter to point to a memory block containing the data that is read from the storage unit; it sets the `_maximum` field to the specified length and the `_length` field to the number of bytes actually read.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODStorageUnit::GetSize` method (page 671).
The `ODStorageUnit::SetValue` method (page 689).

GetWeakStorageUnitRef

The `GetWeakStorageUnitRef` method creates a weak persistent reference to the specified storage unit.

```
void GetWeakStorageUnitRef (
                              in ODStorageUnitID embeddedSUID,
                              out ODStorageUnitRef weakRef);
```

`embeddedSUID`
The storage-unit ID of the storage unit whose persistent reference is desired.

`weakRef` The persistent reference to the storage unit specified by the `embeddedSUID` parameter.

DISCUSSION

Before you call this method, you should focus this storage unit on the value where you want to store the weak persistent reference. After this method executes successfully, call the `SetValue` method to store the resulting persistent reference, returned in the `weakRef` output parameter, into the currently focused value.

IMPORTANT

The scope of a persistent reference is limited to the value in which it was created. If you store the persistent reference in a different value, it will almost certainly not refer to the correct storage unit. ▲

EXCEPTIONS

`kODErrIllegalNullStorageUnitInput` The `embeddedSUID` parameter is null.

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODStorageUnitRef` type (page 873).
The `ODStorageUnit::GetStrongStorageUnitRef` method (page 672).
The `ODStorageUnit::IsWeakStorageUnitRef` method (page 681).
The `ODStorageUnit::SetValue` method (page 689).
For more information on persistent references, see the chapter on storage in the *OpenDoc Programmer's Guide for the Mac OS*.

IncrementGenerationNumber

The `IncrementGenerationNumber` method increments and returns the generation number of the currently focused value.

```
ODULong IncrementGenerationNumber ();
```

return value The generation number of the currently focused value, expressed as an unsigned 32-bit value.

DISCUSSION

You can use the generation number of a value to tell whether the data in the value has changed. For example, when your part makes a signification change to the data in a value, you can call this method to increment its generation number.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::GetGenerationNumber` method (page 664).

InsertValue

The `InsertValue` method inserts data into the currently focused value, starting at the offset (inclusive).

```
void InsertValue (in ODByteArray value);
```

`value` A byte array whose buffer contains the data to be written.

DISCUSSION

You call this method to insert data into the currently focused value without overwriting the existing data at and beyond the current offset. If the focused value is currently a promise value, the promise is fulfilled before the data is written.

This method writes data to the focused value, starting at the current offset. If the focused value contained any data at and beyond the offset, that data appears after the inserted data. The size of the value is automatically increased to accommodate the inserted data.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODStorageUnit::DeleteValue` method (page 656).
The `ODStorageUnit::SetValue` method (page 689).

Internalize

The `Internalize` method reads all properties and values from this storage unit into memory.

```
ODStorageUnit Internalize ();
```

return value A reference to this storage unit.

DISCUSSION

OpenDoc calls this method; your part does not call this method.

EXCEPTIONS

`kODErrInvalidStorageUnit`
This storage unit is not valid.

SEE ALSO

The `ODStorageUnit::Externalize` method (page 660).

IsPromiseValue

The `IsPromiseValue` method returns a Boolean value that indicates whether the currently focused value is a promise value.

```
ODBoolean IsPromiseValue ();
```

return value `kODTrue` if the currently focused value is a promise value, otherwise `kODFalse`.

DISCUSSION

If the currently focused value is a promise value, the promise is not resolved by this method.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::GetPromiseValue` method (page 668).
The `ODStorageUnit::SetPromiseValue` method (page 686).

IsStrongStorageUnitRef

The `IsStrongStorageUnitRef` method returns a Boolean value that indicates whether the specified persistent reference is a strong persistent reference.

```
ODBoolean IsStrongStorageUnitRef (in ODStorageUnitRef ref);
```

`ref` The persistent reference to be tested (assumed to be valid).

return value `kODTrue` if the specified reference is a strong persistent reference, otherwise `kODFalse`.

DISCUSSION

Before calling this method, you can call the `IsValidStorageUnitRef` method to check whether the specified persistent reference is valid.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitRef` type (page 873).
The `ODStorageUnit::GetStrongStorageUnitRef` method (page 672).
The `ODStorageUnit::IsWeakStorageUnitRef` method (page 681).
The `ODStorageUnit::IsValidStorageUnitRef` method (page 680).

IsValidStorageUnitRef

The `IsValidStorageUnitRef` method returns a Boolean value that indicates whether the specified persistent reference is valid.

```
ODBoolean IsValidStorageUnitRef (in ODStorageUnitRef aRef);
```

`aRef` The persistent reference to be tested.

return value `kODTrue` if the specified persistent reference is valid, otherwise `kODFalse`.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a value.

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitRef` type (page 873).

IsWeakStorageUnitRef

The `IsWeakStorageUnitRef` method returns a Boolean value that indicates whether the specified persistent reference is a weak persistent reference.

```
ODBoolean IsWeakStorageUnitRef (in ODStorageUnitRef ref);
```

`ref`	The persistent reference to be tested (assumed to be valid).
return value	`kODTrue` if the specified reference is a weak persistent reference, otherwise `kODFalse`.

DISCUSSION

Before calling this method, you can call the `IsValidStorageUnitRef` method to check whether the specified persistent reference is valid.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitRef` type (page 873).
The `ODStorageUnit::GetWeakStorageUnitRef` method (page 675).
The `ODStorageUnit::IsStrongStorageUnitRef` method (page 679).
The `ODStorageUnit::IsValidStorageUnitRef` method (page 680).

Lock

The `Lock` method locks this storage unit for exclusive access.

```
ODStorageUnitKey Lock (in ODStorageUnitKey key);
```

`key`	The previously acquired identifier for a particular clone operation, or the value `kODNULLKey` if the valid storage unit key is unknown.
return value	The identifier for the locked state of this storage unit.

DISCUSSION

Every thread must acquire the storage unit key for multithreading support.

EXCEPTIONS

`kODErrInvalidStorageUnitKey`	The specified storage unit key is not valid.

SEE ALSO

The `ODStorageUnitKey` type (page 873).
The `ODStorageUnit::Unlock` method (page 690).

Remove

The `Remove` method removes all properties and values in the current focus context from this storage unit.

```
ODStorageUnit Remove ();
```

return value	A reference to this storage unit.

DISCUSSION

If the current focus context is the entire storage unit, this method removes all properties and their values. If it is focused on a property, this method removes the focused property and all its values. If it is focused on a value, this method removes the focused value.

After this method executes successfully, the storage unit is unfocused.

EXCEPTIONS

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::AddProperty` method (page 647).
The `ODStorageUnit::AddValue` method (page 648).

RemoveStorageUnitRef

The `RemoveStorageUnitRef` method makes a persistent reference invalid in the currently focused value.

```
ODStorageUnit RemoveStorageUnitRef (
                                    in ODStorageUnitRef aRef);
```

`aRef`	The persistent reference to be removed.
return value	A reference to this storage unit.

DISCUSSION

This method does not change the data in the currently focused value, but after this method is called, the specified persistent reference is no longer valid. To remove data corresponding to the persistent reference, you must call the `DeleteValue` method.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitRef` type (page 873).
The `ODStorageUnit::DeleteValue` method (page 656).

ResolveAllPromises

The `ResolveAllPromises` method resolves all promises in this storage unit.

```
void ResolveAllPromises ();
```

DISCUSSION

To resolve a promise, the storage unit calls the `FulfillPromise` method of the part that made the promise.

This method does not change the current focus context.

EXCEPTIONS

`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODPart::FulFillPromise` method (page 502).
The `ODStorageUnit::ClearAllPromises` method (page 649).

SetName

The `SetName` method sets the name of this storage unit.

```
void SetName (in ODStorageUnitName name);
```

`name` The name to assign to this storage unit.

EXCEPTIONS

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnitName` type (page 873).
The `ODStorageUnit::GetName` method (page 667).

SetOffset

The `SetOffset` method sets the offset of the currently focused value.

```
void SetOffset (in ODULong offset);
```

`offset` The new offset (in bytes) of the read/write position from the beginning of the data stream in the currently focused value, expressed as an unsigned 32-bit value.

DISCUSSION

You can call this method if you want to read or write data at a particular position in the focused value. An offset of 0 means the beginning of the data stream corresponding to the focused value; an offset equal to the current size of the focused value (as returned by the `GetSize` method) means the end of the data stream. You may not specify an offset larger than the current size of the focused value.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODStorageUnit::GetOffset` method (page 667).
The `ODStorageUnit::GetSize` method (page 671).

SetPromiseValue

The `SetPromiseValue` method writes data to the specified value of the currently focused property, creating the value if it doesn't exist and making the value a promise value.

```
void SetPromiseValue (in ODValueType valueType,
                      in ODULong offset,
                      in ODByteArray value,
                      in ODPart sourcePart);
```

`valueType`	The type of the value where the promise data is to be written.
`offset`	The offset at which the promise data is to be stored, expressed as an unsigned 32-bit number of bytes from the beginning of the value.
`value`	A byte array whose buffer contains the promise data to be written.
`sourcePart`	A reference to the part that made the promise.

DISCUSSION

You call this method to write a promise for a value of the specified type in the currently focused property. You may call this method multiple times to promise values of different types or to write to different offsets in the same value.

This method writes data to the specified value, starting at the specified offset (inclusive), and overwrites any data at and beyond the offset. If the current offset plus the length of data being written is greater than the current size of the value (as returned by the `GetSize` method), the size of the value is automatically increased to accommodate the new data.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrIllegalNullValueTypeInput`
: The specified value type is null.

`kODErrInvalidValueType`
: The specified value type is improperly formed or illegal.

`kODErrUnfocusedStorageUnit`
: This storage unit is not focused on a property or a value.

`kODErrZeroRefCount`
: This storage unit has a reference count of 0.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODValueType` type (page 874).
The `ODStorageUnit::GetPromiseValue` method (page 668).
The `ODStorageUnit::GetSize` method (page 671).
The `ODStorageUnit::IsPromiseValue` method (page 679).

SetStorageUnitRef

The `SetStorageUnitRef` method creates in the currently focused value a persistent reference to the specified storage unit using the specified the persistent identifier.

```
void SetStorageUnitRef (in ODStorageUnitID embeddedSUID,
                        in ODStorageUnitRef ref);
```

`embeddedSUID`
The storage-unit ID for the storage unit to be referenced.

`ref` The persistent identifier for the new persistent reference being created.

DISCUSSION

This method is called only by the container suite. Parts, the document shell, and container applications should not call this method.

The `embeddedSUID` parameter specifies the ID that identifies the storage unit within the current session; the `ref` parameter specifies the reference to be used within the currently focused value to identify the storage unit persistently across sessions.

EXCEPTIONS

`kODErrIllegalNullIDInput`
The `embeddedSUID` parameter is null.

`kODErrUnfocusedStorageUnit`
This storage unit is not focused on a value.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODStorageUnitRef` type (page 873).

SetType

The `SetType` method sets the type of the currently focused value.

```
void SetType (in ODValueType valueType);
```

`valueType` The new type for the currently focused value.

EXCEPTIONS

`kODErrInvalidValueType`	The specified value type is improperly formed or illegal.
`kODErrUnfocusedStorageUnit`	This storage unit is not focused on a value.
`kODErrZeroRefCount`	This storage unit has a reference count of 0.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageUnit::GetType` method (page 673).

SetValue

The `SetValue` method writes data to the currently focused value, starting at the offset (inclusive).

```
void SetValue (in ODByteArray value);
```

`value` A byte array whose buffer contains the data to be written.

DISCUSSION

You call this method to write data to the currently focused value. If that value currently is a promise value, the promise is fulfilled before the data is written.

This method writes data to the focused value, starting at the current offset, and overwrites any data at and beyond the offset. If the current offset plus the length of data being written is greater than the current size of the value (as returned by the `GetSize` method), the size of the value is automatically increased to accommodate the new data.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrUnfocusedStorageUnit` This storage unit is not focused on a value.

`kODErrZeroRefCount` This storage unit has a reference count of 0.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODStorageUnit::GetSize` method (page 671).
The `ODStorageUnit::GetValue` method (page 674).

Unlock

The `Unlock` method unlocks this storage unit.

```
void Unlock (in ODStorageUnitKey key);
```

`key` The storage unit key acquired from the `Lock` method.

DISCUSSION

Every thread must acquire the storage unit key for multithreading support.

EXCEPTIONS

`kODErrInvalidStorageUnitKey` The specified storage unit key is not valid.

`kODErrStorageUnitNotLocked` This storage unit is not locked.

SEE ALSO

The `ODStorageUnitKey` type (page 873).
The `ODStorageUnit::Lock` method (page 682).

ODStorageUnitCursor

Superclasses `ODObject`

Subclasses none

An object of the `ODStorageUnitCursor` class provides swift focusing on frequently accessed data in a storage unit.

Description

A storage-unit cursor represents a **focus context** for a storage unit. The focus context can be the entire storage unit, a particular property, or a particular value of a particular property. When the focus context is the entire storage unit, the data of interest includes all properties and all their values. When the focus context is a particular property, the data of interest includes all values of that property. When the focus context is a particular value, the data of interest is the data stream corresponding to that value.

A storage-unit cursor uses a property name, a value type, and a value index to specify its focus context. Methods of the storage-unit cursor allow you to get and set each of these three pieces of information.

- To specify the entire storage unit as the focus context, set the property name to `kODNULL`; set the value type to `kODTypeAll` and the value index to `kODIndexAll`.
- To specify a property as the focus context, set the property name to the name of the desired property; set the value type to `kODTypeAll` and the value index to `kODIndexAll`.
- To specify a value as the focus context, set the property name to the name of the property containing the value. You can specify the value by either its type or its index:
  - To use type, set the value type to the type of the desired value and the value index to 0.

- To use index, set the value type to `kODNULL` and the value index to the index of the desired value.

A storage-unit cursor makes it simple for you to focus the storage unit on the corresponding focus context. Your part creates a storage-unit cursor object by calling its storage unit's `CreateCursor` (page 653) or `CreateCursorWithFocus` (page 654) methods. Your part calls its storage unit's `FocusWithCursor` method (page 663) to focus the storage unit on the focus context represented by a storage-unit cursor.

For more information about storage units, focus contexts, and value indexes, see the `ODStorageUnit` class description (page 641).

Methods

This section presents summary descriptions of the `ODStorageUnitCursor` methods, followed by detailed descriptions in alphabetical order.

`GetProperty`	Gets the property name of this storage-unit cursor.
`SetProperty`	Sets the property name of this storage-unit cursor.
`GetValueIndex`	Gets the value index of this storage-unit cursor.
`SetValueIndex`	Sets the value index of this storage-unit cursor.
`GetValueType`	Gets the value type of this storage-unit cursor.
`SetValueType`	Sets the value type of this storage-unit cursor.

GetProperty

The `GetProperty` method gets the property name of this storage-unit cursor.

```
void GetProperty (out ODPropertyName propertyName);
```

`propertyName`
The name of the property in the focus context.

DISCUSSION

When you no longer need the property name returned in the `propertyName` parameter, you should deallocate it.

SEE ALSO

The `ODPropertyName` type (page 874).
The `ODStorageUnitCursor::SetProperty` method (page 694).

GetValueIndex

The `GetValueIndex` method gets the value index of this storage-unit cursor.

```
void GetValueIndex (out ODValueIndex valueIndex);
```

`valueIndex` The value index in the focus context.

SEE ALSO

The `ODValueIndex` type (page 874).
The `ODStorageUnitCursor::SetValueIndex` method (page 694).

GetValueType

The `GetValueType` method gets the value type of this storage-unit cursor.

```
void GetValueType (out ODValueType valueType);
```

`valueType` The value type of the focus context.

DISCUSSION

When you no longer need the value type returned in the `valueType` parameter, you should deallocate it.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageUnitCursor::SetValueType` method (page 695).

SetProperty

The `SetProperty` method sets the property name of this storage-unit cursor.

```
void SetProperty (in ODPropertyName propertyName);
```

`propertyName`
The name of the property in the focus context, or `kODNULL` if the focus context is the entire storage unit.

DISCUSSION

When you no longer need the property name you pass as the `propertyName` parameter, you should deallocate it.

SEE ALSO

The `ODPropertyName` type (page 874).
The `ODStorageUnitCursor::GetProperty` method (page 692).

SetValueIndex

The `SetValueIndex` method sets the value index of this storage-unit cursor.

```
void SetValueIndex (in ODValueIndex valueIndex);
```

`valueIndex` The value index of the value in the focus context, or 0 to ignore the value index (and use the value type to specify the value in the focus context).

DISCUSSION

When you use this storage-unit cursor to focus your part's storage unit, the value index of this storage-unit cursor is ignored unless the value type is `kODNULL`.

SEE ALSO

The `ODValueIndex` type (page 874).
The `ODStorageUnitCursor::GetValueIndex` method (page 693).

SetValueType

The `SetValueType` method sets the value type of this storage-unit cursor.

```
void SetValueType (in ODValueType valueType);
```

`valueType` The value type of the value in the focus context, or `kODNULL` to ignore the value type (and use the value index to specify the value in the focus context).

DISCUSSION

When you no longer need the value type you pass as the `valueType` parameter, you should deallocate it.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageUnitCursor::GetValueType` method (page 693).

ODStorageUnitRefIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODStorageUnitRefIterator` class provides access to all persistent references in a currently focused value.

Description

If a value in a storage unit contains persistent references, a storage-unit reference iterator can provide access to each of those persistent references. For example, a caller might need to access all referenced storage units in a clone operation. A caller can also use a storage-unit reference iterator to access the persistent reference in any storage unit value. This practice might be helpful for utilities that fix up cross-references or index a document. Persistent references cannot be created or removed during iteration, and the storage unit must remain focused to the value on which the storage-unit reference iterator is iterating.

Callers create a storage-unit reference iterator object by calling its storage unit's `CreateStorageUnitRefIterator` method (page 655), which returns a reference to a storage-unit reference iterator object.

For more information on cloning, see the chapters on storage and data transfer in the *OpenDoc Programmer's Guide for the Mac OS*. For more information on accessing objects through iterators, see its chapter on OpenDoc runtime features.

Methods

This section presents summary descriptions of the `ODStorageUnitRefIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and obtains a copy of the first persistent reference, if it exists, in the currently focused value of the storage unit.
`Next`	Obtains a copy of the next persistent reference, if it exists, in the currently focused value of the storage unit.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and obtains a copy of the first persistent reference, if it exists, in the currently focused value of the storage unit.

```
void First (out ODStorageUnitRef ref);
```

`ref`	A copy of the first persistent reference in the currently focused value of the storage unit. If the focused value contains no persistent references, the return value is undefined.

DISCUSSION

Your part must call this method before calling the `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

It is your responsibility to deallocate the returned persistent reference when it is no longer needed.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The focused value was modified while the iteration was in progress.

SEE ALSO

The `ODStorageUnitRef` type (page 873).

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more persistent references remain in the focused value. This method returns `kODTrue` if the preceding call to the `First`, `Last`, `Next`, or `Previous` method found a persistent reference. This method returns `kODFalse` when you have examined all the persistent references.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The focused value was modified while the iteration was in progress.

Next

The `Next` method obtains a copy of the next persistent reference, if it exists, in the currently focused value of the storage unit.

```
void Next (out ODStorageUnitRef ref);
```

`ref` A copy of the next persistent reference in the currently focused value of the storage unit. If the focused value contains no persistent references or if the iteration is complete, the return value is undefined.

DISCUSSION

If your part calls this method before calling this storage-unit reference iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

It is your responsibility to deallocate the returned persistent reference when it is no longer needed.

EXCEPTIONS

`kODErrIteratorOutOfSync` The focused value was modified while the iteration was in progress.

SEE ALSO

The `ODStorageUnitRef` type (page 873).

ODStorageUnitView

Superclasses `ODObject`

Subclasses none

An object of the `ODStorageUnitView` class provides thread-safe access to a storage unit by automatically focusing and locking the storage unit.

Description

A storage-unit view represents a particular storage unit, prefocused on a particular focus context. Your part creates a storage-unit view object by focusing a storage unit then calling the `CreateView` method (page 656) of that storage unit. You can pass the storage-unit view among your software components to give them access to the particular focused data stream.

A storage-unit view has most of the functionality of a storage unit, except that it has no methods for changing the focus. When you access a storage unit through a storage-unit view, however, the storage-unit view automatically locks the storage unit during the access and unlocks it afterward.

The storage-unit view has an associated storage-unit cursor that represents the focus context of the storage-unit view. You can call the `GetCursor` method (page 710) of a storage-unit view to obtain its storage-unit cursor. If you make changes to that storage-unit cursor, you change the focus context of the storage-unit view. Typically, however, parts do not change the focus context of a storage-unit view.

For more information related to storage units and storage-unit cursors, see the description of the classes `ODStorageUnit` (page 641) and `ODStorageUnitCursor` (page 691).

Methods

This section presents summary descriptions of the `ODStorageUnitView` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Storage

`CloneInto`	Copies to the specified destination storage unit all properties and values of the storage unit that created this storage-unit view.
`Externalize`	Resolves all promises in the storage unit that created this storage-unit view and saves all its properties and values to persistent, external storage.
`Internalize`	Reads into memory all properties and values from the storage unit that created this storage-unit view.

Storage Unit Manipulation

`GetStorageUnit`	Returns a reference to the storage unit that created this storage-unit view.
`AddProperty`	Adds a property with the specified name to the storage unit that created this storage-unit view.
`AddValue`	Adds a value of the specified type to the focused property.
`Remove`	Removes all properties and values in the focus context from the storage unit that created this storage-unit view.
`GetID`	Returns the storage-unit ID for the storage unit that created this storage-unit view.
`GetName`	Returns the name of the storage unit that created this storage-unit view.
`SetName`	Sets the name of the storage unit that created this storage-unit view.
`GetSize`	Returns the size (number of bytes) of the data in the focus context.

Focus Context Access

`GetCursor` Returns a reference to the storage-unit cursor representing the focus context of this storage-unit view.

`GetProperty` Returns the name of the property in the focus context.

Value Manipulation

`GetValue` Reads data from the focused value, starting at the offset (inclusive).

`SetValue` Writes data to the focused value, starting at the offset (inclusive).

`DeleteValue` Deletes data from the focused value, starting at the offset (inclusive).

`InsertValue` Inserts data into the focused value, starting at the offset (inclusive).

`GetOffset` Returns the offset of the focused value.

`SetOffset` Sets the offset of the focused value.

`GetType` Returns the type of the focused value.

`SetType` Sets the type of the focused value.

`GetGenerationNumber` Returns the generation number of the focused value.

`IncrementGenerationNumber` Increments and returns the generation number of the focused value.

Persistent Reference Manipulation

`GetIDFromStorageUnitRef` Returns the storage-unit ID of a referenced storage unit.

`GetStrongStorageUnitRef` Creates a strong persistent reference to the specified storage unit.

`GetWeakStorageUnitRef` Creates a weak persistent reference to the specified storage unit.

`IsStrongStorageUnitRef`
: Returns a Boolean value that indicates whether the specified persistent reference is a strong persistent reference.

`IsWeakStorageUnitRef`
: Returns a Boolean value that indicates whether the specified persistent reference is a weak persistent reference.

`IsValidStorageUnitRef`
: Returns a Boolean value that indicates whether the specified persistent reference is valid.

`RemoveStorageUnitRef`
: Makes a persistent reference invalid in the focused value.

Promise Manipulation

`IsPromiseValue`	Returns a Boolean value that indicates whether the focused value is a promise value.
`GetPromiseValue`	Reads promise data from the specified value of the focused property.
`SetPromiseValue`	Writes data to the specified value of the focused property, creating the value if it doesn't exist and making the value a promise value.

Creating Objects

`CreateStorageUnitRefIterator`
: Creates a storage-unit reference iterator for the focused value.

AddProperty

The `AddProperty` method adds a property with the specified name to the storage unit that created this storage-unit view.

```
ODStorageUnitView AddProperty (
                            in ODPropertyName propertyName);
```

`propertyName`
The name of the property to be added.

return value A reference to this storage-unit view.

DISCUSSION

If the storage unit that created this storage-unit view does not already contain a property with the specified name, the new property is added and the storage unit is focused on the newly added property. Otherwise, the storage unit is focused on the existing property with the specified name.

The focus context of this storage-unit view remains unchanged.

EXCEPTIONS

`kODErrCannotAddProperty`
Cannot add the specified property to the storage unit that created this storage-unit view.

`kODErrIllegalNullPropertyInput`
The specified property name is null.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODPropertyName` type (page 874).
The `ODStorageUnitView::GetProperty` method (page 716).
The `ODStorageUnitView::Remove` method (page 729).

AddValue

The `AddValue` method adds a value of the specified type to the focused property.

```
ODStorageUnitView AddValue (in ODValueType type);
```

`type`	The type of value to be added.
return value	A reference to this storage-unit view.

DISCUSSION

If the focused property does not already contain a value of the specified type, the new value is added and the storage unit that created this storage-unit view is focused on the newly added value. Otherwise, the storage unit is focused on the existing value with the specified value type.

The focus context of this storage-unit view remains unchanged.

EXCEPTIONS

`kODErrIllegalNullValueTypeInput`
: The specified value type is null.

`kODErrInvalidValueType`
: The specified value type is improperly formed or illegal.

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a property or a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageUnitView::Remove` method (page 729).

CloneInto

The `CloneInto` method copies to the specified destination storage unit all properties and values of the storage unit that created this storage-unit view.

```
void CloneInto (in ODDraftKey key,
                in ODStorageUnit destStorageUnit,
                in ODID scopeID);
```

`key` The draft key identifying this cloning operation.

`destStorageUnit`
A reference to the destination storage unit to which the data is to be copied.

`scopeID` The ID of the frame that defines the scope of this cloning operation.

DISCUSSION

This method is not called by parts. Your part should call its draft's `Clone` or `WeakClone` method instead of this method.

The storage unit that created this storage-unit view is the source storage unit for the cloning operation.

If the source storage unit has persistent references to other objects, the `scopeID` parameter determines which of the referenced objects are within the scope of this cloning operation. Typically, the `scopeID` parameter is the ID of a frame, and only those objects embedded in that frame are within scope. In the rare case in which the `scopeID` parameter is `kODIDAll`, all referenced objects are within scope.

This method copies data from the source storage unit into the specified destination storage unit. If the source storage unit has persistent references to other objects, this method clones any persistently referenced objects that are within the scope of this cloning operation. Objects reference by strong persistent references are strongly cloned by recursive calls to the `Clone` method; objects referenced by weak persistent references are weakly cloned by calls to the `WeakClone` method.

EXCEPTIONS

`kODErrInvalidDraftKey`
: The specified draft key is not the draft key for the current cloning transaction.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODDraftKey` type (page 872).
The `ODID` type (page 869).
The `ODDraft::Clone` method (page 159).
The `ODDraft::WeakClone` method (page 181).

CreateStorageUnitRefIterator

The `CreateStorageUnitRefIterator` method creates a storage-unit reference iterator for the focused value.

```
ODStorageUnitRefIterator CreateStorageUnitRefIterator ();
```

return value A reference to a storage-unit reference iterator used to traverse all persistent references in the focused value.

DISCUSSION

You can call this method if you need to perform an operation on all storage-unit references in the focused value.

While you are using the returned storage-unit reference iterator, you must not modify the focused value; in particular, you must not call any of the following methods of this storage-unit view: `DeleteValue`, `Internalize`, `Remove`, `RemoveStorageUnitRef`, `SetType`, or `SetValue`. Furthermore, you must not delete the storage unit that created this storage-unit view.

You must delete the returned storage-unit reference iterator when it is no longer needed.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitRefIterator` class (page 696).

DeleteValue

The `DeleteValue` method deletes data from the focused value, starting at the offset (inclusive).

```
void DeleteValue (in ODULong length);
```

`length`
: The number of bytes of data to delete, expressed as an unsigned 32-bit value.

DISCUSSION

You call this method to delete data from the focused value. If that value is a promise value, the promise is fulfilled before the data is deleted. This method starts deleting data at the current offset and stops after deleting the number of bytes specified by the `length` parameter or after reaching the end of the data in the currently focused value, whichever comes first.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitView::InsertValue` method (page 724).
The `ODStorageUnitView::Remove` method (page 729).

Externalize

The `Externalize` method resolves all promises in the storage unit that created this storage-unit view and saves all its properties and values to persistent, external storage.

```
ODStorageUnitView Externalize ();
```

return value A reference to this storage-unit view.

EXCEPTIONS

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitView::Internalize` method (page 725).

GetCursor

The `GetCursor` method returns a reference to the storage-unit cursor representing the focus context of this storage-unit view.

```
ODStorageUnitCursor GetCursor ();
```

return value A reference to the storage-unit cursor representing the focus context of this storage-unit view.

DISCUSSION

You can change the focus context of this storage-unit view by modifying the returned storage-unit cursor; typically, however, parts do not change the focus context of a storage-unit view.

You should not delete the returned storage-unit cursor.

GetGenerationNumber

The `GetGenerationNumber` method returns the generation number of the focused value.

```
ODULong GetGenerationNumber ();
```

return value The generation number of the focused value, expressed as an unsigned 32-bit value.

DISCUSSION

You can use the generation number of a value to tell whether the data in the value has changed. For example, your part could compare the number returned by this method with a saved generation number.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitView::IncrementGenerationNumber` method (page 723).

GetID

The `GetID` method returns the storage-unit ID for the storage unit that created this storage-unit view.

```
ODID GetID ();
```

return value The storage-unit ID for the storage unit that created this storage-unit view.

SEE ALSO

The `ODID` type (page 869).
The `ODStorageUnitView::GetIDFromStorageUnitRef` method (page 712).

GetIDFromStorageUnitRef

The `GetIDFromStorageUnitRef` method returns the storage-unit ID of a referenced storage unit.

```
ODStorageUnitID GetIDFromStorageUnitRef (
                                    in ODStorageUnitRef aRef);
```

`aRef`	A persistent storage-unit reference.
return value	The storage-unit ID the storage unit referenced by the `aRef` parameter.

DISCUSSION

The focused value must be the same value that created the specified persistent reference. This method looks up the storage unit referenced by the specified persistent reference and returns its storage-unit ID.

EXCEPTIONS

`kODErrInvalidStorageUnitRef`
: The specified persistent reference is not valid.

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODStorageUnitRef` type (page 873).
The `ODStorageUnitView::GetID` method (page 711).

GetName

The `GetName` method returns the name of the storage unit that created this storage-unit view.

```
ODStorageUnitName GetName ();
```

return value The name of the storage unit that created this storage-unit view, or `kODNULL` if the storage unit does not have a name.

DISCUSSION

When you no longer need the returned name, you should deallocate it.

SEE ALSO

The `ODStorageUnitName` type (page 873).
The `ODStorageUnitView::SetName` method (page 731).

GetOffset

The `GetOffset` method returns the offset of the focused value.

```
ODULong GetOffset ();
```

return value The offset (in bytes) of the read/write position from the beginning of the data stream in the focused value, expressed as an unsigned 32-bit value.

DISCUSSION

An offset of 0 means the beginning of the data stream corresponding to the focused value; an offset equal to the size returned by the `GetSize` method means the end of the data stream.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitView::GetSize` method (page 717).
The `ODStorageUnitView::SetOffset` method (page 731).

GetPromiseValue

The `GetPromiseValue` method reads promise data from the specified value of the focused property.

```
ODULong GetPromiseValue (in ODValueType valueType,
                         in ODULong offset,
                         in ODULong length,
                         out ODByteArray value,
                         out ODPart sourcePart);
```

`valueType` The type of the value from which the promise data is to be read.

`offset` The offset from which the promise data is to be retrieved, expressed as an unsigned 32-bit number of bytes from the beginning of the value.

`length` The length (in bytes) of the data to be retrieved, expressed as an unsigned 32-bit value.

`value` A byte array structure to contain the retrieved promise data.

`sourcePart` A reference to the part that made the promise.

return value The number of bytes read, expressed as an unsigned 32-bit value.

DISCUSSION

You call this method to read promise data without fulfilling the promise. This method first focuses the storage unit on the specified value of the focused property. It then starts reading data at the specified offset and stops after reading the number of bytes specified by the `length` parameter or after reaching the end of the data in the focused value, whichever comes first.

When you call this method, the `_buffer` field of the `value` output parameter should be `kODNULL`; if it isn't, the buffer to which that field points will not be deallocated.

This method sets the `_buffer` field of the `value` output parameter to point to a memory block containing the promise data, the `_maximum` field to the specified length, and the `_length` field to the number of bytes actually read.

This method sets the `sourcePart` output parameter to a reference to the part that made the promise. This method does not increment the reference count of the part that made the promise.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrInvalidValueType`
: The specified value type is improperly formed or illegal.

`kODErrSUValueDoesNotExist`
: The focused property does not have a value with the specified value type.

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a property or a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODValueType` type (page 874).
The `ODStorageUnitView::IsPromiseValue` method (page 725).
The `ODStorageUnitView::SetPromiseValue` method (page 732).

GetProperty

The `GetProperty` method returns the name of the property in the focus context.

```
ODPropertyName GetProperty ();
```

return value The name of the property in the focus context.

DISCUSSION

When you no longer need the returned property name, you should deallocate it.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a property or a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODPropertyName` type (page 874).
The `ODStorageUnitView::AddProperty` method (page 703).
The `ODStorageUnitView::Remove` method (page 729).

GetSize

The `GetSize` method returns the size (number of bytes) of the data in the focus context.

```
ODULong GetSize ();
```

return value The size (in bytes) of the data in the focus context, expressed as an unsigned 32-bit value.

DISCUSSION

If the focus context is the storage unit that created this storage-unit view, this method returns the total size of all properties and values in the storage unit. If the focus context is a property, this method returns the total size of all values in the focused property. If the focus context is a value, this method returns the size of the data stream corresponding to the focused value.

If the focused value is a promise value, the promise is fulfilled before the size of the value is evaluated.

EXCEPTIONS

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

GetStorageUnit

The `GetStorageUnit` method returns a reference to the storage unit that created this storage-unit view.

```
ODStorageUnit GetStorageUnit ();
```

return value A reference to the storage unit that created this storage-unit view.

DISCUSSION

This method does not increment the reference count of the returned storage unit. For that reason, if you cache the returned storage unit, you should call its `Acquire` method to increment its reference count and then call its `Release` method when you are finished using it.

GetStrongStorageUnitRef

The `GetStrongStorageUnitRef` method creates a strong persistent reference to the specified storage unit.

```
void GetStrongStorageUnitRef (
                              in ODStorageUnitID embeddedSUID,
                              out ODStorageUnitRef strongRef);
```

`embeddedSUID`
: The storage-unit ID of the storage unit whose persistent reference is desired.

`strongRef`
: The persistent reference to the storage unit specified by the `embeddedSUID` parameter.

DISCUSSION

After this method executes successfully, call the `SetValue` method to store the resulting persistent reference, returned in the `strongRef` output parameter, into the focused value.

IMPORTANT

The scope of a persistent reference is limited to the value in which it was created. If you store the persistent reference in a different value, it will almost certainly not refer to the correct storage unit. ▲

EXCEPTIONS

`kODErrIllegalNullStorageUnitInput`
: The `embeddedSUID` parameter is null.

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODStorageUnitRef` type (page 873).
The `ODStorageUnitView::GetWeakStorageUnitRef` method (page 721).
The `ODStorageUnitView::IsStrongStorageUnitRef` method (page 726).
The `ODStorageUnitView::SetValue` method (page 735).
For more information on persistent references, see the chapter on storage in the *OpenDoc Programmer's Guide for the Mac OS*.

GetType

The `GetType` method returns the type of the focused value.

```
ODValueType GetType ();
```

return value The type of the focused value.

DISCUSSION

When you no longer need the returned value type, you should deallocate it.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageUnitView::SetType` method (page 734).

GetValue

The `GetValue` method reads data from the focused value, starting at the offset (inclusive).

```
ODULong GetValue (in ODULong length,
                  out ODByteArray value);
```

`length`	The length (number of bytes) of data to read, expressed as an unsigned 32-bit value.
`value`	A byte array structure to contain the retrieved data.
return value	The number of bytes read, expressed as an unsigned 32-bit value.

DISCUSSION

You call this method to read data from the focused value. If that value is a promise value, the promise is fulfilled before the data is read. This method starts reading data at the current offset and stops after reading the number of bytes specified by the `length` parameter or after reaching the end of the data in the currently focused value, whichever comes first.

When you call this method, the `_buffer` field of the `value` output parameter should be `kODNULL`; if it isn't, the buffer to which that field points will not be deallocated.

This method sets the `_buffer` field of the `value` output parameter to point to a memory block containing the data that is read from the storage unit; it sets the `_maximum` field to the specified length and the `_length` field to the number of bytes actually read.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODStorageUnitView::GetSize` method (page 717).
The `ODStorageUnitView::SetValue` method (page 735).

GetWeakStorageUnitRef

The `GetWeakStorageUnitRef` method creates a weak persistent reference to the specified storage unit.

```
void GetWeakStorageUnitRef (
                            in ODStorageUnitID embeddedSUID,
                            out ODStorageUnitRef weakRef);
```

`embeddedSUID`
: The storage-unit ID of the storage unit whose persistent reference is desired.

`weakRef`
: The persistent reference to the storage unit specified by the `embeddedSUID` parameter.

DISCUSSION

After this method executes successfully, call the `SetValue` method to store the resulting persistent reference, returned in the `weakRef` output parameter, into the focused value.

IMPORTANT

The scope of a persistent reference is limited to the value in which it was created. If you store the persistent reference in a different value, it will almost certainly not refer to the correct storage unit. ▲

EXCEPTIONS

`kODErrIllegalNullStorageUnitInput`
: The `embeddedSUID` parameter is null.

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitID` type (page 873).
The `ODStorageUnitRef` type (page 873).
The `ODStorageUnitView::GetStrongStorageUnitRef` method (page 718).
The `ODStorageUnitView::IsWeakStorageUnitRef` method (page 728).
The `ODStorageUnitView::SetValue` method (page 735).

For more information on persistent references, see the chapter on storage in the *OpenDoc Programmer's Guide for the Mac OS*.

IncrementGenerationNumber

The `IncrementGenerationNumber` method increments and returns the generation number of the focused value.

```
ODULong IncrementGenerationNumber ();
```

return value The generation number of the focused value, expressed as an unsigned 32-bit value.

DISCUSSION

You can use the generation number of a value to tell whether the data in the value has changed. For example, when your part makes a signification change to the data in a value, you can call this method to increment its generation number.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitView::GetGenerationNumber` method (page 710).

InsertValue

The `InsertValue` method inserts data into the focused value, starting at the offset (inclusive).

```
void InsertValue (in ODByteArray value);
```

`value` A byte array whose buffer contains the data to be written.

DISCUSSION

You call this method to insert data into the focused value without overwriting the existing data at and beyond the current offset. If the focused value is currently a promise value, the promise is fulfilled before the data is written.

This method writes data to the focused value, starting at the current offset. If the focused value contained any data at and beyond the offset, that data appears after the inserted data. The size of the value is automatically increased to accommodate the inserted data.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODStorageUnitView::DeleteValue` method (page 708).
The `ODStorageUnitView::SetValue` method (page 735).

Internalize

The `Internalize` method reads into memory all properties and values from the storage unit that created this storage-unit view.

```
ODStorageUnitView Internalize ();
```

return value A reference to this storage-unit view.

DISCUSSION

OpenDoc calls this method; your part does not call this method.

SEE ALSO

The `ODStorageUnitView::Externalize` method (page 709).

IsPromiseValue

The `IsPromiseValue` method returns a Boolean value that indicates whether the focused value is a promise value.

```
ODBoolean IsPromiseValue ();
```

return value `kODTrue` if the focused value is a promise value, otherwise `kODFalse`.

DISCUSSION

If the focused value is a promise value, the promise is not resolved by this method.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitView::GetPromiseValue` method (page 714).
The `ODStorageUnitView::SetPromiseValue` method (page 732).

IsStrongStorageUnitRef

The `IsStrongStorageUnitRef` method returns a Boolean value that indicates whether the specified persistent reference is a strong persistent reference.

```
ODBoolean IsStrongStorageUnitRef (in ODStorageUnitRef ref);
```

`ref`	The persistent reference to be tested (assumed to be valid).
return value	`kODTrue` if the specified reference is a strong persistent reference, otherwise `kODFalse`.

DISCUSSION

Before calling this method, you can call the `IsValidStorageUnitRef` method to check whether the specified persistent reference is valid.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitRef` type (page 873).
The `ODStorageUnitView::GetStrongStorageUnitRef` method (page 718).
The `ODStorageUnitView::IsWeakStorageUnitRef` method (page 728).
The `ODStorageUnitView::IsValidStorageUnitRef` method (page 727).

IsValidStorageUnitRef

The `IsValidStorageUnitRef` method returns a Boolean value that indicates whether the specified persistent reference is valid.

```
ODBoolean IsValidStorageUnitRef (in ODStorageUnitRef ref);
```

`ref`	The persistent reference to be tested.
return value	`kODTrue` if the specified persistent reference is valid, otherwise `kODFalse`.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitRef` type (page 873).

IsWeakStorageUnitRef

The `IsWeakStorageUnitRef` method returns a Boolean value that indicates whether the specified persistent reference is a weak persistent reference.

```
ODBoolean IsWeakStorageUnitRef (in ODStorageUnitRef ref);
```

`ref`	The persistent reference to be tested (assumed to be valid).
return value	`kODTrue` if the specified reference is a weak persistent reference, otherwise `kODFalse`.

DISCUSSION

Before calling this method, you can call the `IsValidStorageUnitRef` method to check whether the specified persistent reference is valid.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitRef` type (page 873).
The `ODStorageUnitView::GetWeakStorageUnitRef` method (page 721).
The `ODStorageUnitView::IsStrongStorageUnitRef` method (page 726).
The `ODStorageUnitView::IsValidStorageUnitRef` method (page 727).

Remove

The `Remove` method removes all properties and values in the focus context from the storage unit that created this storage-unit view.

```
ODStorageUnitView Remove ();
```

return value A reference to this storage-unit view.

DISCUSSION

If the focus context of this storage-unit view is the entire storage unit, this method removes all properties and their values. If the focus context is a property, this method removes the focused property and all its values. If the focus context is a value, this method removes the focused value.

After this method executes successfully, the storage unit that created this storage-unit view is unfocused. The focus context of this storage-unit view is unchanged.

This method should be used with caution; if it executes successfully, it makes the focus context of this storage-unit view invalid.

EXCEPTIONS

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitView::AddProperty` method (page 703).
The `ODStorageUnitView::AddValue` method (page 705).

RemoveStorageUnitRef

The `RemoveStorageUnitRef` method makes a persistent reference invalid in the focused value.

```
ODStorageUnitView RemoveStorageUnitRef (
                                    in ODStorageUnitRef aRef);
```

`aRef` The persistent reference to be removed.

return value A reference to this storage-unit view.

DISCUSSION

This method does not change the data in the focused value, but after this method is called, the specified persistent reference is no longer valid. To remove data corresponding to the persistent reference, you must call the `DeleteValue` method.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitRef` type (page 873).
The `ODStorageUnitView::DeleteValue` method (page 708).

SetName

The `SetName` method sets the name of the storage unit that created this storage-unit view.

```
void SetName (in ODStorageUnitName name);
```

`name` The name to assign to the storage unit that created this storage-unit view.

SEE ALSO

The `ODStorageUnitName` type (page 873).
The `ODStorageUnitView::GetName` method (page 713).

SetOffset

The `SetOffset` method sets the offset of the focused value.

```
void SetOffset (in ODULong offset);
```

`offset` The new offset (in bytes) of the read/write position from the beginning of the data stream in the focused value, expressed as an unsigned 32-bit value.

DISCUSSION

You can call this method if you want to read or write data at a particular position in the focused value. An offset of 0 means the beginning of the data stream corresponding to the focused value; an offset equal to the current size of the focused value (as returned by the `GetSize` method) means the end of the data stream. You may not specify an offset larger than the current size of the focused value.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODStorageUnitView::GetOffset` method (page 713).
The `ODStorageUnitView::GetSize` method (page 717).

SetPromiseValue

The `SetPromiseValue` method writes data to the specified value of the focused property, creating the value if it doesn't exist and making the value a promise value.

```
void SetPromiseValue (in ODValueType valueType,
                      in ODULong offset,
                      in ODByteArray value,
                      in ODPart sourcePart);
```

`valueType` The type of the value where the promise data is to be written.

`offset` The offset at which the promise data is to be stored, expressed as an unsigned 32-bit number of bytes from the beginning of the value.

`value` A byte array whose buffer contains the promise data to be written.

`sourcePart` A reference to the part that made the promise.

DISCUSSION

You call this method to write a promise for a value of the specified type in the focused property. You may call this method multiple times to promise values of different types or to write to different offsets in the same value.

This method writes data to the specified value, starting at the specified offset (inclusive), and overwrites any data at and beyond the offset. If the current offset plus the length of data being written is greater than the current size of the value (as returned by the `GetSize` method), the size of the value is automatically increased to accommodate the new data.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrIllegalNullValueTypeInput`
: The specified value type is null.

`kODErrInvalidValueType`
: The specified value type is improperly formed or illegal.

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a property or a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODValueType` type (page 874).
The `ODStorageUnitView::GetPromiseValue` method (page 714).
The `ODStorageUnitView::GetSize` method (page 717).
The `ODStorageUnitView::IsPromiseValue` method (page 725).

SetType

The `SetType` method sets the type of the focused value.

```
void SetType (in ODValueType valueType);
```

`valueType` The new type of the focused value.

DISCUSSION

This method should be used with caution; it may make this storage-unit view invalid.

EXCEPTIONS

`kODErrInvalidValueType`
: The specified value type is improperly formed or illegal.

`kODErrUnfocusedStorageUnit`
: The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageUnitView::GetType` method (page 719).

SetValue

The `SetValue` method writes data to the focused value, starting at the offset (inclusive).

```
void SetValue (in ODByteArray value);
```

`value` A byte array whose buffer contains the data to be written.

DISCUSSION

You call this method to write data to the focused value. If that value currently is a promise value, the promise is fulfilled before the data is written.

This method writes data to the focused value, starting at the current offset, and overwrites any data at and beyond the offset. If the current offset plus the length of data being written is greater than the current size of the value (as returned by the `GetSize` method), the size of the value is automatically increased to accommodate the new data.

When you no longer need the structure you pass as the `value` parameter, you should deallocate that structure and its buffer.

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
The focus context of this storage-unit view is not a value.

If the storage-unit cursor for this storage-unit view does not represent a legal focus context for the storage unit that created this storage-unit view, this method throws exceptions raised by the `FocusWithCursor` method of that storage unit.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODStorageUnitView::GetSize` method (page 717).
The `ODStorageUnitView::GetValue` method (page 720).

ODTransform

Superclasses `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODTransform` class maps points and shapes from one coordinate system to another. This can be viewed as modifying a shape, for example, by scaling or rotating it.

Description

A transform uses a 3-by-3 matrix to perform two-dimensional transformations. The simplest transform is an identity transform, which has no effect on any points or shapes that it transforms.

Your part creates a new identity transform by calling the `CreateTransform` method (page 310) of a frame, the `CreateTransform` method (page 237) of a facet, or the `NewTransform` method (page 752) of an existing transform. Your part can create a copy of an existing transform by calling that transform's `Copy` method (page 740).

You can use a transform to perform the following transformations on coordinates or shapes:

- Translation (offset) shifts the position of a shape.
- Scaling changes the size of a shape.
- Rotation changes the angle of rotation of a shape, rotating all points around a given point.
- Skewing changes the slant applied to a shape.
- Perspective modifies the positions of points to give a three-dimensional effect.

For more information on matrices and transformations in two-dimensional drawing, you can consult any standard computer-graphics textbook, such as *Computer Graphics Principles and Practice,* second edition, by Foley, vanDam,

Feiner, and Hughes (Addison-Wesley, 1990). You can also refer to the chapter on drawing in the *OpenDoc Programmer's Guide for the Mac OS*.

You do not need to subclass `ODTransform`. However, you can provide for new transforms by creating subclasses of `ODTransform`. For example, you can define new transforms that do not use transformation matrices. These new transforms might perform more complex operations, such as morphing or wrapping around a 3D surface.

Overriding Inherited Methods

The following methods are inherited and available for use by your subclass of `ODTransform`.

somInit

The `somInit` method initializes the instance variables in a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODTransform`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should initialize the new instance variables in this transform object. The SOM library calls this method when this transform object is created. You must not do anything that might fail in this method. This limits you to operations such as setting pointer variables to null, setting numeric variables to appropriate values, and making similar assignments from constants. If you have any initialization code that can potentially fail, it must be handled in this transform object's subclass-specific initialization method; see also the `InitTransform` method (page 747).

somUninit

The `somUninit` method disposes of the storage created for a SOM object; it is inherited from the `SOMObject` class.

If you subclass `ODTransform`, you can override this method. Your override method does not need to call its inherited method; the inherited method is automatically called for you by the SOM library.

Your override of this method should dispose of any storage created for this transform object, including any storage related to additional instance variables initialized in this transform object. The SOM library calls this method when this transform object is deleted; this method must not fail.

Purge

The `Purge` method frees memory on request; it is inherited from the `ODObject` class.

```
ODSize Purge (in ODSize size);
```

Every subclass of `ODObject` can override this method and should do so if it creates caches and temporary buffers. If you subclass `ODTransform`, you must override this method or risk running out of available memory. Your override of this method must call its inherited method at some point in your implementation (it does not matter where). You should save the size value returned by the inherited method because you will need it to compute the value to return from your override method.

Your override of this method should free any caches, noncritical buffers, or objects (up to the amount of memory specified). Your override of this method should add the number of bytes actually freed to the number returned by the inherited method and return the sum as the total amount of memory released. OpenDoc calls this method in low-memory situations; you should not allocate memory for this operation.

Methods

This section presents summary descriptions of the `ODTransform` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [M] are specific to the Mac OS platform.

Creating Transforms

`NewTransform`	Creates a new identity transform.
`Copy`	Creates a new transform that is a copy of this transform.

Applying the Transform

`TransformShape`	Modifies the specified shape by applying this transform.
`InvertShape`	Modifies the specified shape by applying the inverse of this transform.
`TransformPoint`	Modifies the location of the specified point by applying this transform.
`InvertPoint`	Modifies the location of the specified point by applying the inverse of this transform's matrix.

Testing the Transform

`GetType`	Returns the transform type of this transform.
`IsSameAs`	Returns a Boolean value that indicates whether this transform is identical to the specified transform.
`HasMatrix`	Returns a Boolean value that indicates whether this transform uses a matrix to describe its transformation.

Manipulating the Matrix

`ReadFrom`	Reads this transform's matrix from the specified storage unit.
`WriteTo`	Writes this transform's matrix to the specified storage unit.
`GetMatrix`	Copies this transform's matrix into the specified structure.
`SetMatrix`	Replaces this transform's matrix with the specified matrix.
`PostCompose`	Modifies this transform's matrix by postmultiplying it with the specified transform's matrix.
`PreCompose`	Modifies this transform's matrix by premultiplying it with the specified transform's matrix.
`CopyFrom`	Modifies this transform to make it equivalent to the specified source transform.
`Invert`	Inverts this transform's matrix.
`Reset`	Changes this transform to an identity transform.

Manipulating the Translation Values

`GetOffset`	Returns this transform's translation values (also called offset values) in the specified structure.
`SetOffset`	Changes this transform into a pure offset with the specified horizontal and vertical translation values.
`GetPreScaleOffset`	Returns, in the specified structure, the offset to use if you are going to apply the offset before scaling.
`MoveBy`	Offsets this transform's horizontal and vertical translation values by the specified amount.
`IsQDOffset` [M]	Returns a Boolean value that indicates whether this transform is a pure offset with integral translation values.
`GetQDOffset` [M]	Returns this transform's offset value expressed as a QuickDraw point.
`SetQDOffset` [M]	Changes this transform into a pure offset with the specified integral horizontal and vertical translation values.

Manipulating the Scaling Factors

`GetScale`	Returns this transform's horizontal and vertical scaling factors in the specified structure.
`ScaleBy`	Scales this transform by the specified vertical and horizontal scaling factors.
`ScaleDownBy`	Scales this transform by the reciprocal of the specified vertical and horizontal scaling factors.

Initializing

`InitTransform`	Initializes this transform. This method is needed only if you subclass `ODTransform`.

Copy

The `Copy` method creates a new transform that is a copy of this transform.

```
ODTransform Copy ();
```

return value A reference to the newly created transform.

DISCUSSION

The new transform does not share any data with this transform; therefore, you can modify each of the transforms independently.

This method initializes the reference count of the returned transform. When you have finished using that transform, you should call its `Release` method.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

EXCEPTION

`kODErrOutOfMemory`	There is not enough memory to copy the transform.

CopyFrom

The `CopyFrom` method modifies this transform to make it equivalent to the specified source transform.

```
ODTransform CopyFrom (in ODTransform sourceTransform);
```

`sourceTransform`
A reference to the source transform to be copied.

return value A reference to this transform modified to be a copy of the specified transform.

DISCUSSION

After this method executes successfully, this transform and the source transform do not share data; therefore, you can modify each of the transforms independently.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

GetMatrix

The `GetMatrix` method copies this transform's matrix into the specified structure.

```
void GetMatrix (out ODMatrix matrix);
```

`matrix` The structure in which to return the matrix.

DISCUSSION

If you use transforms of the class `ODTransform` and you have also created a subclass of `ODTransform` that applies complex transformation effects that cannot be represented by matrices, your part needs to check whether a given transform has a matrix before calling this method. To do so, you should call the transform's `HasMatrix` method; only if that method returns true should you call the transform's `GetMatrix` method.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method must not call its inherited method. Instead, it should raise a `kODErrTransformErr` exception. You can detect that a transform of your class has no matrix by calling its `HasMatrix` method.

EXCEPTION

`kODErrTransformErr`	This transform object does not use a matrix.

SEE ALSO

The `ODMatrix` type (page 858).
The `ODTransform::HasMatrix` method (page 746).
The `ODTransform::SetMatrix` method (page 756).

GetOffset

The `GetOffset` method returns this transform's translation values (also called offset values) in the specified structure.

```
void GetOffset (out ODPoint offset);
```

`offset`	A point specifying the horizontal (x) and vertical (y) translation values of this transform.

DISCUSSION

The returned horizontal and vertical translation values are located in this transform's matrix; they are the first two elements of the bottom row in the matrix.

SEE ALSO

The `ODPoint` type (page 855).
The `ODTransform::GetPreScaleOffset` method (page 744).
The `ODTransform::SetOffset` method (page 757).

GetPreScaleOffset

The `GetPreScaleOffset` method returns, in the specified structure, the offset to use if you are going to apply the offset before scaling.

```
void GetPreScaleOffset (out ODPoint offset);
```

`offset` A point specifying the horizontal (x) and vertical (y) translation values to use before scaling.

DISCUSSION

This method is useful if you want to transform your data by first offsetting and then scaling it. It should not be used if the transformation involves rotation, skewing, or perspective.

SEE ALSO

The `ODPoint` type (page 855).
The `ODTransform::GetOffset` method (page 743).

GetQDOffset

Mac OS

The `GetQDOffset` method returns this transform's offset value expressed as a QuickDraw point.

```
Point GetQDOffset ();
```

return value A QuickDraw point that represents this transform's offset value. The horizontal and vertical translation values are rounded to the nearest integer.

SEE ALSO

The `ODTransform::SetQDOffset` method (page 758).

GetScale

The `GetScale` method returns this transform's horizontal and vertical scaling factors in the specified structure.

```
void GetScale (out ODPoint scale);
```

`scale` A point specifying the horizontal (x) and vertical (y) scaling factors.

DISCUSSION

Two elements in the transform's matrix specify the amount by which a shape is scaled. The horizontal scaling factor is determined by the element in the top-left corner of the matrix. The vertical scaling factor is governed by the element in the center of the matrix.

SEE ALSO

The `ODPoint` type (page 855).

GetType

The `GetType` method returns the transform type of this transform.

```
ODTransformType GetType ();
```

return value The transform type of this transform.

DISCUSSION

The returned value specifies the transform's function, which is one of the following:

- Identity transform (`kODIdentityXform`).
- Pure translation or offset (`kODTranslateXform`).
- Pure scale (`kODScaleXform`).
- Scale and translation (`kODScaleTranslateXform`).
- Scale, rotate, and skew (`kODLinearXform`).
- Scale, rotate, skew, and translation (`kODLinearTranslateXform`).
- Perspective transformation, which applies a 3D or distortion effect (`kODPerspectiveXform`).

HasMatrix

The `HasMatrix` method returns a Boolean value that indicates whether this transform uses a matrix to describe its transformation.

```
ODBoolean HasMatrix ();
```

return value `kODTrue` if the transform object uses a matrix, otherwise `kODFalse`.

DISCUSSION

Every object of the `ODTransform` class uses a matrix; hence this method returns true. However, if you use transforms of the class `ODTransform` and you have also created a subclass of `ODTransform` that applies complex transformation effects that cannot be represented by matrices, you can call this method to test whether a particular transform belongs to a class that uses a transform matrix.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method must not call its inherited method. Instead, it should return `kODFalse`.

InitTransform

The `InitTransform` method initializes this transform. This method is needed only if you subclass `ODTransform`.

```
void InitTransform ();
```

DISCUSSION

This method is not called directly to initialize this transform object, but is called by a subclass-specific initialization method. By convention, every subclass of `ODTransform` should have a separate initialization method (for example, the `InitMyTransform` method) that is called when an instance of that subclass is created. The initialization method may have additional parameters beyond those of the `InitTransform` method. The `InitMyTransform` method should call the inherited `InitTransform` method at the beginning of its implementation.

If you subclass `ODTransform`, your subclass-specific initialization method, rather than its `somInit` method, should handle any initialization code that can potentially fail. For example, your initialization method may attempt to allocate memory for your transform.

OVERRIDING

If you subclass `ODTransform`, you should not override this method.

Invert

The `Invert` method inverts this transform's matrix.

```
ODTransform Invert ();
```

return value A reference to this transform with its matrix changed to represent the inverse of its original matrix.

DISCUSSION

The inverse of a transform is the mathematical inverse of its matrix. It has the exact opposite geometric effect of the original transform.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to invert the matrix.
`kODErrTransformErr`	The transform has no inverse. This is not true of most real-world transformations, only for those that perform transformations such as flattening a shape into a single line or point.

InvertPoint

The `InvertPoint` method modifies the location of the specified point by applying the inverse of this transform matrix.

```
void InvertPoint (inout ODPoint point);
```

`point` The point to be modified. On return, the fields of this structure have been modified to represent the modified point.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

EXCEPTIONS

`kODErrOutOfMemory` There is not enough memory to compute the inverse matrix.

SEE ALSO

The `ODPoint` type (page 855).
The `ODTransform::TransformPoint` method (page 759).

InvertShape

The `InvertShape` method modifies the specified shape by applying the inverse of this transform.

```
void InvertShape (in ODShape shape);
```

`shape` A reference to the shape object whose geometric representation is to be modified by the inverse of this transform.

DISCUSSION

This method is operationally equivalent to the `InverseTransform` method of the specified shape.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

SEE ALSO

The `ODShape::InverseTransform` method (page 619).
The `ODTransform::TransformShape` method (page 759).

IsQDOffset

Mac OS

The `IsQDOffset` method returns a Boolean value that indicates whether this transform is a pure offset with integral translation values.

```
ODBoolean IsQDOffset ();
```

return value `kODTrue` if this transform is a pure offset with integral translation values, otherwise `kODFalse`.

DISCUSSION

A transform that is a pure offset with integral values can be represented exactly by a QuickDraw point.

SEE ALSO

The `ODTransform::GetQDOffset` method (page 744).
The `ODTransform::SetQDOffset` method (page 758).

IsSameAs

The `IsSameAs` method returns a Boolean value that indicates whether this transform is identical to the specified transform.

```
ODBoolean IsSameAs (in ODTransform compareTransform);
```

`compareTransform`
A reference to the transform to be used for comparison.

return value `kODTrue` if the transforms are equivalent (that is, they describe the same transformation), otherwise `kODFalse`.

DISCUSSION

The transform objects are equal if, and only if, their matrices are identical or if one is an exact multiple of the other.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

MoveBy

The `MoveBy` method offsets this transform's horizontal and vertical translation values by the specified amount.

```
ODTransform MoveBy (in ODPoint point);
```

`point` A point specifying the horizontal (x) and vertical (y) translations to be added to the current translation values.

return value A reference to this transform after the offset operation.

SEE ALSO

The `ODPoint` type (page 855).

NewTransform

The `NewTransform` method creates a new identity transform.

```
ODTransform NewTransform ();
```

return value A reference to the newly created transform, or `kODNULL` if an error occurred.

DISCUSSION

This method initializes the reference count of the returned transform. When you have finished using that transform, you should call its `Release` method.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to create a new transform.

PostCompose

The `PostCompose` method modifies this transform's matrix by postmultiplying it with the specified transform's matrix.

```
ODTransform PostCompose (in ODTransform transform);
```

`transform` A reference to the transform whose matrix is to be postmultiplied with this transform's matrix.

return value A reference to this transform after the postcompose operation.

DISCUSSION

Postcomposing multiplies this transform's matrix on the right side by the specified transform:

```
this ← this × transform
```

The resulting transform has the same effect as applying the two original transforms in sequence: first this transform, then the other.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

PreCompose

The `PreCompose` method modifies this transform's matrix by premultiplying it with the specified transform's matrix.

```
ODTransform PreCompose (in ODTransform transform);
```

`transform` A reference to the transform whose matrix is to be premultiplied with this transform's matrix.

return value A reference to this transform after the precompose operation.

DISCUSSION

Precomposing multiplies this transform's matrix on the left side by the specified transform.

```
this ← transform × this
```

The resulting transform has the same effect as applying the two original transforms in sequence: first the other transform, then this one.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

ReadFrom

The `ReadFrom` method reads this transform's matrix from the specified storage unit.

```
void ReadFrom (in ODStorageUnit storageUnit);
```

`storageUnit`
A reference to the storage unit from which the matrix is to be read.

DISCUSSION

Before calling this method, you must focus the storage unit on the property from which the matrix elements are to be written. The matrix is read from the value of type `kODTransform` in the focused property. If no such value exists, this transform is reset to an identity transform.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
This storage unit is not focused on a property or a value.

SEE ALSO

The `ODTransform::WriteTo` method (page 760).

Reset

The `Reset` method changes this transform to an identity transform.

```
ODTransform Reset ();
```

return value A reference to this transform reset to the identity transform.

DISCUSSION

Except for transforms created by the `Copy` method, newly created transforms start out as identity transforms; this method changes a transform back to the initial state.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

ScaleBy

The `ScaleBy` method scales this transform by the specified vertical and horizontal scaling factors.

```
ODTransform ScaleBy (in ODPoint scale);
```

`scale` A point specifying the horizontal (x) and vertical (y) scaling factors.

return value A reference to this transform after the scaling operation.

SEE ALSO

The `ODPoint` type (page 855).
The `ODTransform::ScaleDownBy` method (page 756).

ScaleDownBy

The `ScaleDownBy` method scales this transform by the reciprocal of the specified vertical and horizontal scaling factors.

```
ODTransform ScaleDownBy (in ODPoint scale);
```

`scale`	A point specifying the horizontal (x) and vertical (y) scaling factors.
return value	A reference to this transform after the scaling operation.

DISCUSSION

This method is the inverse of the transform's `ScaleBy` method.

SEE ALSO

The `ODPoint` type (page 855).
The `ODTransform::ScaleBy` method (page 755).

SetMatrix

The `SetMatrix` method replaces this transform's matrix with the specified matrix.

```
ODTransform SetMatrix (in ODMatrix matrix);
```

`matrix`	The new transform matrix for this transform.
return value	A reference to this transform with it matrix replaced.

DISCUSSION

If you specify an identity matrix, the effect of this method is the same as calling the `Reset` method.

If you use transforms of the class `ODTransform` and you have also created a subclass of `ODTransform` that applies complex transformation effects that cannot be represented by matrices, your part needs to check whether a given transform has a matrix before calling this method. To do so, you should call the transform's `HasMatrix` method; only if that method returns true should you call the transform's `SetMatrix` method.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method must not call its inherited method. Instead, it should raise a `kODErrTransformErr` exception. You can detect that a transform of your class has no matrix by calling its `HasMatrix` method.

EXCEPTION

`kODErrTransformErr`	This transform object does not use a matrix.

SEE ALSO

The `ODMatrix` type (page 858).
The `ODTransform::GetMatrix` method (page 742).
The `ODTransform::HasMatrix` method (page 746).
The `ODTransform::Reset` method (page 755).

SetOffset

The `SetOffset` method changes this transform into a pure offset with the specified horizontal and vertical translation values.

```
ODTransform SetOffset (in ODPoint point);
```

`point`	A point specifying the new horizontal (x) and vertical (y) translation values.
return value	A reference to this transform changed to an offset transform.

DISCUSSION

The translation values are the first two elements of the bottom row of a transform's matrix.

SEE ALSO

The `ODPoint` type (page 855).
The `ODTransform::GetOffset` method (page 743).

SetQDOffset

Mac OS

The `SetQDOffset` method changes this transform into a pure offset with the specified integral horizontal and vertical translation values.

```
ODTransform SetQDOffset (in Point point);
```

`point`	A QuickDraw point that represents the new offset for this transform.
return value	A reference to this transform changed to an offset transform with integral translation values.

DISCUSSION

The specified translation values are integer values.

SEE ALSO

The `ODTransform::GetQDOffset` method (page 744).

TransformPoint

The `TransformPoint` method modifies the location of the specified point by applying this transform.

```
void TransformPoint (inout ODPoint point);
```

`point` The point to be modified. On return, the fields of this structure have been modified to represent the transformed point.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

SEE ALSO

The `ODPoint` type (page 855).
The `ODTransform::InvertPoint` method (page 748).

TransformShape

The `TransformShape` method modifies the specified shape by applying this transform.

```
void TransformShape (in ODShape shape);
```

`shape` A reference to the shape object to be modified.

DISCUSSION

Calling this method is equivalent to calling the `Transform` method of the specified shape.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to transform the shape.

SEE ALSO

The `ODShape::Transform` method (page 631).
The `ODTransform::InvertShape` method (page 749).

WriteTo

The `WriteTo` method writes this transform's matrix to the specified storage unit.

```
void WriteTo (in ODStorageUnit storageUnit);
```

`storageUnit`
A reference to the storage unit where this transform's matrix is to be written.

DISCUSSION

Before calling this method, you must focus the storage unit to the property where the matrix is to be written. This method writes the matrix into the value of type `kODTransform` in the focused property, replacing any matrix that was previously stored in that value or creating the value if it doesn't already exist.

OVERRIDING

If you subclass `ODTransform` to create a nonlinear transform class, you must override this method. Your override method can call its inherited method at any point in your implementation (it does not matter where).

EXCEPTIONS

`kODErrUnfocusedStorageUnit`
: This storage unit is not focused on a property or a value.

SEE ALSO

The `ODTransform::ReadFrom` method (page 754).

ODTranslation

Superclasses `ODObject`

Subclasses none

An object of the `ODTranslation` class provides data translation services for OpenDoc documents and their parts.

Description

The `ODTranslation` class depends on platform-specific system services to provide OpenDoc data translation. OpenDoc uses translation objects to maintain information on what kinds of translations are available to the user. OpenDoc and part editors can also use the translation object to perform any requested translations, rather than directly calling the underlying platform-specific services.

When a document is opened, the session object creates a single translation object. All parts of the document share the translation object; you can obtain a reference to it by calling the session object's `GetTranslation` method (page 587).

The translation service can be triggered when a part does not know how to handle data of an unfamiliar type (for example, when a user initiates linking or importing data from clipboard or drag-and-drop objects). The part can ask the translation object to translate the data into a recognizable format (part kind). Similarly, OpenDoc can initiate translation when opening a part on a system in which no part editor can directly read the part's data.

OpenDoc does not use platform types (for example, `OSType` on the Mac OS). Therefore, to interoperate between OpenDoc and non-OpenDoc systems, platform types and ISO types need to be translated from one kind to the other. If you have part data expressed as a Mac OS file type and need to express it as a part kind (ISO string), you can use the translation object's `GetISOTypeFromPlatformType` method (page 764) to find out if there is a part kind equivalent to that file type. To convert in the opposite direction, use

the translation object's `GetPlatformTypeFromISOType` method (page 766) instead.

For more information related to data translation, see the chapter on data transfer in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODTranslation` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Translation Availability

`GetTranslationOf` Returns a type list to which the specified value type can be translated.

`CanTranslate` Indicates whether translation is possible from the specified value type.

Translating

`Translate` Tries to translate the specified source data to the specified value type.

`TranslateView` Translates the content of the source storage-unit view and stores the translated data in the destination storage-unit view.

Type Conversion

`GetISOTypeFromPlatformType`
Returns the ISO type corresponding to the specified platform-specific type.

`GetPlatformTypeFromISOType`
Returns the platform-specific type corresponding to the specified ISO type.

CanTranslate

The `CanTranslate` method indicates whether translation is possible from the specified value type.

```
ODTranslateResult CanTranslate (in ODValueType fromType);
```

`fromType`	The type of data to be translated.
return value	The result of the translation. The return value is one of the following: `kODCanTranslate` or `kODCannotTranslate`.

DISCUSSION

The return value `kODCanTranslate` indicates that translation is allowed with the specified type. The return value `kODCannotTranslate` indicates that translation is not allowed with the specified type.

Your part calls this method to determine if translation facilities are available for a particular value type.

SEE ALSO

The `ODTranslateResult` type (page 890).
The `ODValueType` type (page 874).

GetISOTypeFromPlatformType

The `GetISOTypeFromPlatformType` method returns the ISO type corresponding to the specified platform-specific type.

```
ODValueType GetISOTypeFromPlatformType (
                        in ODPlatformType platformType,
                        in ODPlatformTypeSpace typeSpace);
```

`platformType`
: A 32-bit wrapper for the platform-specific type. On the Mac OS platform, this type is identical to the `ScrapType` or `OSType` types (a four-character code).

`typeSpace`
: A 32-bit value used to specify the type of a platform-specific structure identifying a type space (data or file). The value of `typeSpace` must be one of the following: `kODPlatformDataType` or `kODPlatformFileType`.

return value
: The corresponding ISO type.

DISCUSSION

The value `kODPlatformDataType` for the `typeSpace` parameter indicates the native operating system scrap type. The value `kODPlatformFileType` indicates the native operating system file type.

Your part calls this method. OpenDoc does not use platform types (for example, `OSType` on the Mac OS). Therefore, to interoperate between OpenDoc and non-OpenDoc systems, platform types and ISO types need to be translated from one kind to the other.

It is your responsibility to deallocate the returned value type when it is no longer needed.

SEE ALSO

The `ODPlatformTypeSpace` type (page 890).
The `ODValueType` type (page 874).
The `ODStorageSystem::CreatePlatformTypeList` method (page 638).
The `ODTranslation::GetPlatformTypeFromISOType` method (page 766).

GetPlatformTypeFromISOType

The `GetPlatformTypeFromISOType` method returns the platform-specific type corresponding to the specified ISO type.

```
ODPlatformType GetPlatformTypeFromISOType (
                                        in ODValueType type);
```

`type`	The ISO type.
return value	A 32-bit wrapper for the corresponding platform-specific type. On the Mac OS platform, this type is identical to the `ScrapType` or `OSType` types (a four-character code).

DISCUSSION

Your part calls this method. OpenDoc does not use platform types (for example, `OSType` on the Mac OS). Therefore, to interoperate between OpenDoc and non-OpenDoc systems, platform types and ISO types need to be translated from one kind to the other.

SEE ALSO

The `ODValueType` type (page 874).
The `ODStorageSystem::CreatePlatformTypeList` method (page 638).
The `ODTranslation::GetISOTypeFromPlatformType` method (page 764).

GetTranslationOf

The `GetTranslationOf` method returns a type list to which the specified value type can be translated.

```
ODTypeList GetTranslationOf (in ODValueType fromType);
```

`fromType`	The type of data to be translated.

return value A reference to a type list specifying a set of part kinds to which the specified value type can be translated, or an empty list if the translation cannot be achieved.

DISCUSSION

Your part calls this method to determine all possible results that you can obtain by translating the specified type of data. This method does not change the value of the data to be translated.

EXCEPTIONS

`kODErrNoSysTranslationFacility` The underlying system translation facility is not present.

`kODErrOutOfMemory` There is not enough memory to allocate the type list object.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODValueType` type (page 874).

Translate

The `Translate` method tries to translate the specified source data to the specified value type.

```
ODTranslateResult Translate (in ODValueType fromType,
                             in ODByteArray fromData,
                             in ODValueType toType,
                             out ODByteArray toData);
```

`fromType` The type of the source data to be translated.

`fromData`	A byte array whose buffer contains the source data to be translated.
`toType`	The type to which the source data is to be translated.
`toData`	A byte array whose buffer is to contain the translated data.
return value	The result of the translation. The return value is one of the following: `kODCanTranslate` or `kODCannotTranslate`.

DISCUSSION

The return value `kODCanTranslate` indicates that translation is allowed with the specified type. The return value `kODCannotTranslate` indicates that translation is not allowed with the specified type.

Your part calls this method after calling the `CanTranslate` method to establish that the source type is translatable. This method does not change the content of the source byte array.

If translation is successful, this method allocates the destination byte array structure and its buffer, and stores the translated data in that buffer. It is your responsibility to deallocate the byte array structure (and its buffer) when it is no longer needed.

EXCEPTIONS

`kODErrNoSysTranslationFacility`	The underlying system translation facility is not present.
`kODErrOutOfMemory`	There is not enough memory to allocate the destination byte array structure (or its buffer).

This method may throw platform-specific exceptions.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODTranslateResult` type (page 890).
The `ODValueType` type (page 874).
The `ODTranslation::CanTranslate` method (page 764).

TranslateView

The `TranslateView` method translates the content of the source storage-unit view and stores the translated data in the destination storage-unit view.

```
ODTranslateResult TranslateView (
                            in ODStorageUnitView fromView,
                            in ODStorageUnitView toView);
```

`fromView`	A reference to a storage-unit view; the focused value that contains the translated data.
`toView`	A reference to a storage-unit view; the focused value that is to contain the translated data.
return value	The result of the translation. The return value is one of the following: `kODCanTranslate` or `kODCannotTranslate`.

DISCUSSION

The return value `kODCanTranslate` indicates that translation is allowed with the specified type. The return value `kODCannotTranslate` indicates that translation is not allowed with the specified type.

Your part calls this method when it wants to translate some data in a storage unit.

EXCEPTIONS

`kODErrNoSysTranslationFacility`	The underlying system translation facility is not present.
`kODErrOutOfMemory`	There is not enough memory to allocate the translated data.

This method may throw platform-specific exceptions.

SEE ALSO

The `ODTranslateResult` type (page 890).

ODTypeList

Superclasses `ODObject`
Subclasses none

An object of the `ODTypeList` class is an ordered set of `ODType` elements.

Description

A type list is an ordered set of elements, each specifying a different value. Elements are of the `ODType` type (page 846) and so are strings that can specify part kinds, focus types, or storage-unit types. Most often, all elements of a type list are part kinds. Because an `ODType` value is a pointer to an ISO string, any value to be added to a type list is copied first. That is, the pointer itself is not added to the list; instead, the string is copied and a pointer to the new copy is added to the list.

To create a type list, call the `CreateTypeList` method (page 639) of the storage-system object. If the call to that method specifies an existing type list, the new type list is initialized to contain a copy of each element in that list. The elements in the new list are in the same order as the elements of the original list. Otherwise, the new list is initialized to an empty list (a list with no elements).

You can add elements one at a time to the end of the type list. OpenDoc ensures that each element of a type list is unique; if you attempt to add a value that is already in the list, the list remains unchanged. You can remove elements from the list, test whether the list contains a particular element, and get the number of elements in the list. If you need to perform an operation for each element of the list, you can create an object of the `ODTypeListIterator` class (page 775), and use it to iterate through the list.

Methods

This section presents summary descriptions of the `ODTypeList` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Manipulating Elements

`AddLast`	Adds an element to the end of this type list.
`Remove`	Removes the specified element from this type list.

Testing

`Contains`	Returns a Boolean value that indicates whether this type list contains the specified element.
`Count`	Returns the number of elements in this type list.

Creating an Iterator

`CreateTypeListIterator`	Creates a type-list iterator for this type list.

AddLast

The `AddLast` method adds an element to the end of this type list.

```
void AddLast (in ODType type);
```

`type` The element to be added to the list.

DISCUSSION

If this type list already contains an element equal to the specified element, no action is taken. Otherwise, a copy of the specified element is added to the end of the list.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to add the specified element to this type list.

SEE ALSO

The `ODType` type (page 846).
The `ODTypeList::Contains` method (page 772).
The `ODTypeList::Remove` method (page 774).

Contains

The `Contains` method returns a Boolean value that indicates whether this type list contains the specified element.

```
ODBoolean Contains (in ODType type);
```

`type`	The element to be tested for inclusion in this list.
return value	`kODTrue` if this type list contains an element equal to the specified element, otherwise `kODFalse`.

SEE ALSO

The `ODType` type (page 846).
The `ODTypeList::AddLast` method (page 771).
The `ODTypeList::Remove` method (page 774).

Count

The `Count` method returns the number of elements in this type list.

```
ODULong Count ();
```

return value The number of elements in this type list, expressed as an unsigned 32-bit value, or 0 if the list is empty.

CreateTypeListIterator

The `CreateTypeListIterator` method creates a type-list iterator for this type list.

```
ODTypeListIterator CreateTypeListIterator ();
```

return value A reference to the newly created type-list iterator.

DISCUSSION

You call this method if you need to apply an operation to each element of this type list.

While you are using the returned type-list iterator, you must not modify this type list; in particular, you must not add or remove elements and you must not delete this list type list.

You must delete the returned type-list iterator when it is no longer needed.

EXCEPTIONS

`kODErrOutOfMemory` There is not enough memory to create the iterator.

SEE ALSO

The `ODTypeListIterator` class (page 775).

Remove

The `Remove` method removes the specified element from this type list.

```
void Remove (in ODType type);
```

`type` The element to be removed.

DISCUSSION

If this type list contains an element equal to the specified element, that element is removed from the list; if not, no action is taken.

SEE ALSO

The `ODType` type (page 846).
The `ODTypeList::AddLast` method (page 771).
The `ODTypeList::Contains` method (page 772).

ODTypeListIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODTypeListIterator` class provides access to each element of a type list.

Description

You use a type-list iterator to apply an operation to each element of a type list. For example, a part might use a type-list iterator to enumerate the part kinds that the part supports and write to storage a representation for each one.

Methods of the `ODTypeListIterator` class return copies of the elements in the type list. This design prevents you from accidentally changing the content of the type list. When you use a type-list iterator, be sure to delete each copied string to avoid a memory leak.

Your part creates a type-list iterator object by calling the type list object's `CreateTypeListIterator` method (page 773), which returns a reference to a type-list iterator object.

While you are using a type-list iterator, you should not modify or delete the type list that created it. You must postpone adding elements to or removing elements from the type list until after you have deleted the iterator.

For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODTypeListIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and returns a copy of the first element in the type list that created this type-list iterator.
`Next`	Returns a copy of the next element in the type list that created this type-list iterator.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and returns a copy of the first element in the type list that created this type-list iterator.

```
ODType First ();
```

return value A copy of the first element in the type list, or `kODNULL` if the type list is empty.

DISCUSSION

Your part must call this method before calling this type-list iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

It is your responsibility to deallocate the returned type value when it is no longer needed.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The type list was modified while the iteration was in progress.

`kODErrOutOfMemory`
: There is not enough memory to create the type.

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more elements remain in the type list. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found an element. This method returns `kODFalse` when you have examined all the elements. If the type list that created this iterator is empty, this method always returns `kODFalse`.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The type list was modified while the iteration was in progress.

Next

The `Next` method returns a copy of the next element in the type list that created this type-list iterator.

```
ODType Next ();
```

return value A copy of the next element in the type list, or `kODNULL` if you have reached the end of the type list.

DISCUSSION

If your part calls this method before calling this type-list iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

It is your responsibility to deallocate the returned type value when it is no longer needed.

EXCEPTIONS

`kODErrIteratorOutOfSync`
The type list was modified while the iteration was in progress.

`kODErrOutOfMemory`
There is not enough memory to create the type.

ODUndo

Superclasses `ODObject`

Subclasses none

An object of the `ODUndo` class holds command history information to support the undo capability—the ability to reverse the effects of recently executed commands—of OpenDoc.

Description

Your part editor stores undoable actions in, and retrieves them from, the undo object. When a document is opened, the session object creates a single undo object. All parts of that document share the undo object; you can obtain a reference to it by calling the session object's `GetUndo` method (page 588).

The undo object contains an undo stack and a redo stack. When an undoable action is performed, the part involved places **action data**—information provided by the part that allows it to reverse the effects of an undoable action—on the undo stack. OpenDoc stores the action data in the undo object's **action history**—the cumulative set of reversible actions available at any one time. When an action needs to be undone, OpenDoc pops the action data from the undo stack onto the redo stack. At the same time, OpenDoc notifies your part so that your part can undo the recently executed action using the stored action data. When an undone action needs to be redone, OpenDoc pops the action data from the redo stack back onto the undo stack. OpenDoc notifies your part so that your part can redo the recently undone action using the stored action data. The undo and redo stacks can be cleared upon a part's request. When clearing is needed, OpenDoc asks each part to dispose of its action data stored in the stacks. The order of disposal is from newer actions to older actions.

There are times when an **action subhistory**—a subset of reversible actions available at any one time—is useful. For example, you may need a new action context when entering a modal state, that is, when your part displays a modal dialog box. OpenDoc allows marks to be placed on the stacks. If a mark is placed on a stack, the stack may be cleared only to the mark. For example,

when the modal dialog box closes, any actions done within the context of the modal dialog box are disposed of. However, all the actions executed before the modal dialog box appeared are preserved in the stacks.

Methods

This section presents summary descriptions of the `ODUndo` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are called only by the document shell or container applications.

Action History Manipulation

`AddActionToHistory` Pushes the action data and its associated part onto the undo stack.

`AbortCurrentTransaction` Removes the current transaction (and any nested transactions) from the action history.

`ClearActionHistory` Clears the undo and redo stacks.

`MarkActionHistory` Marks the top of the undo and redo stacks.

Undo

`Undo` [D] Undoes the top action in the undo stack.

`PeekUndoHistory` Returns a Boolean value that indicates whether there is anything on the undo stack and gets information about the action at the top of the undo stack.

Redo

`Redo` [D] Redoes the top action in the redo stack.

`ClearRedoHistory` Clears the redo stack.

`PeekRedoHistory` Returns a Boolean value that indicates whether there is anything on the redo stack and gets information about the action at the top of the redo stack.

AbortCurrentTransaction

The `AbortCurrentTransaction` method removes the current transaction (and any nested transactions) from the action history.

```
void AbortCurrentTransaction ();
```

DISCUSSION

This method aborts a transaction that is being placed in the undo stack by removing all single actions up to and including the last begin action. If there is a nested transaction in the current transaction, it is entirely removed. This method in turn calls your part's `UndoAction` method to give your part the opportunity to perform any reverse editing necessary to restore itself to the state it possessed before the transaction began.

SEE ALSO

The `ODPart::UndoAction` method (page 528).

AddActionToHistory

The `AddActionToHistory` method pushes the action data and its associated part onto the undo stack.

```
void AddActionToHistory (in ODPart whichPart,
                         in ODActionData actionData,
                         in ODActionType actionType,
                         in ODName undoActionLabel,
                         in ODName redoActionLabel);
```

`whichPart` A reference to the part that performed the action.

`actionData` A byte array whose buffer contains the data needed by the part to allow it to undo the action.

`actionType` The possible type of undo action. The value of `actionType` must be one of the following: `kODSingleAction`, `kODBeginAction`, or `kODEndAction`.

`undoActionLabel`
A user-visible label for the undo command beginning with the word Undo.

`redoActionLabel`
A user-visible label for the redo command beginning with the word Redo.

DISCUSSION

The value `kODSingleAction` for the `actionType` parameter indicates that the action was a single action. The value `kODBeginAction` indicates that the action was the first action of a multistep action. The value `kODEndAction` indicates that the action was the last action of a multistep action.

EXCEPTIONS

`kODErrCannotAddAction`	Cannot add the specified action to this undo object; an undo or redo action is already in progress.
`kODErrOutOfMemory`	There is not enough memory to allocate the action information.

SEE ALSO

The `ODActionData` type (page 868).
The `ODActionType` type (page 868).
The `ODName` type (page 846).

ClearActionHistory

The `ClearActionHistory` method clears the undo and redo stacks.

```
void ClearActionHistory (
                  in ODRespectMarksChoices respectMarks);
```

`respectMarks`
The possible values for clearing an action history. The value of `respectMarks` must be one of the following: `kODRespectMarks` or `kODDontRespectMarks`.

DISCUSSION

The value `kODRespectMarks` for the `respectMarks` parameter indicates that the stacks are cleared down only to the specified marks; that is, only actions within an action subhistory are cleared. The value `kODDontRespectMarks` indicates that the stacks are cleared in their entirety.

ClearRedoHistory

The `ClearRedoHistory` method clears the redo stack.

```
void ClearRedoHistory ();
```

DISCUSSION

OpenDoc calls this method. If the redo stack contains a mark indicating an action subhistory, this method clears only that subhistory. Otherwise, it clears the entire redo stack.

MarkActionHistory

The `MarkActionHistory` method marks the top of the undo and redo stacks.

```
void MarkActionHistory ();
```

DISCUSSION

The marks are used to indicate the beginning of a new action subhistory in each stack.

EXCEPTIONS

`kODErrCannotMarkAction`
: Failure to start an action subhistory by placing a mark at the beginning of the undo and redo stacks; the undo object was not initialized properly.

PeekRedoHistory

The `PeekRedoHistory` method returns a Boolean value that indicates whether there is anything on the redo stack and gets information about the action at the top of the redo stack.

```
ODBoolean PeekRedoHistory (out ODPart part,
                           out ODActionData actionData,
                           out ODActionType actionType,
                           out ODName actionLabel);
```

`part`
: A reference to the part that performed the action at the top of the redo stack.

`actionData`
: A byte array whose buffer is to contain the action data for the action at the top of the redo stack.

`actionType`
: The possible type of undo action. The value of `actionType` must be one of the following: `kODSingleAction`, `kODBeginAction`, or `kODEndAction`.

`actionLabel`
: A user-visible label for the redo command beginning with the word Redo.

return value
: `kODTrue` if there is anything on the redo stack, otherwise `kODFalse`.

DISCUSSION

The value `kODSingleAction` for the `actionType` parameter indicates that the action was a single action. The value `kODBeginAction` indicates that the action was the first action of a multistep action. The value `kODEndAction` indicates that the action was the last action of a multistep action.

The document shell or container applications call this method to properly set up the Redo item. Your part can also call this method, but it is probably unnecessary.

If the top of the redo stack contains a mark indicating an action subhistory, this method returns `kODFalse`.

SEE ALSO

The `ODActionData` type (page 868).
The `ODActionType` type (page 868).
The `ODName` type (page 846).

PeekUndoHistory

The `PeekUndoHistory` method returns a Boolean value that indicates whether there is anything on the undo stack and gets information about the action at the top of the undo stack.

```
ODBoolean PeekUndoHistory (out ODPart part,
                           out ODActionData actionData,
                           out ODActionType actionType,
                           out ODName actionLabel);
```

`part` A reference to the part that performed the action at the top of the undo stack.

`actionData`
A byte array whose buffer is to contain the action data for the action at the top of the undo stack.

`actionType`
The possible type of undo action. The value of `actionType` must be one of the following: `kODSingleAction`, `kODBeginAction`, or `kODEndAction`.

`actionLabel`
A user-visible label for the undo command beginning with the word Undo.

return value `kODTrue` if there is anything on the undo stack, otherwise `kODFalse`.

DISCUSSION

The value `kODSingleAction` for the `actionType` parameter indicates that the action was a single action. The value `kODBeginAction` indicates that the action was the first action of a multistep action. The value `kODEndAction` indicates that the action was the last action of a multistep action.

The document shell or container applications call this method to properly set up the Undo item. Your part can also call this method, but it is probably unnecessary.

If the top of the undo stack contains a mark indicating an action subhistory, this method returns `kODFalse`.

SEE ALSO

The `ODActionData` type (page 868).
The `ODActionType` type (page 868).
The `ODName` type (page 846).

Redo

Document shell

The `Redo` method redoes the top action in the redo stack.

```
void Redo ();
```

EXCEPTIONS

`kODErrEmptyStack`	The redo stack is empty; the undo object was not initialized properly.

This method may throw Apple event exceptions or any other exceptions returned by the part.

Undo

Document shell

The `Undo` method undoes the top action in the undo stack.

```
void Undo ();
```

EXCEPTIONS

`kODErrEmptyStack`	The undo stack is empty; the undo object was not initialized properly.

This method may throw Apple event exceptions or any other exceptions returned by the part.

ODValueIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODValueIterator` class provides access to the entries of a value name space.

Description

You use a value iterator to apply an operation to all entries of a value name space. For example, you might use a value iterator to iterate over a list of part editors to obtain information about each part editor, such as the part's supported part kinds.

Your part creates a value iterator object by calling the value name space's `CreateIterator` method (page 794), which returns a reference to a value iterator object. The iterator performs an unordered traversal of the name space.

While you are using a value iterator, you should not modify or delete the name space. You must postpone adding entries to or removing entries from the name space until after you have deleted the iterator.

For more information related to value name spaces, see the `ODValueNameSpace` class description (page 793). For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODValueIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and obtains the first entry in the name space.
`Next`	Obtains the next entry in the name space.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and obtains the first entry in the name space.

```
void First (out ODISOStr key,
            out ODByteArray value);
```

`key`	A pointer to an ISO string representing the key in the first entry in the name space, or `kODNULL` for an empty name space.
`value`	A byte array whose buffer contains the value for the first entry in the name space.

DISCUSSION

Your part must call this method before calling this value iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

It is your responsibility to deallocate the ISO string and byte array structure (and its buffer) when they are no longer needed. You must also delete the key when it is no longer needed.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The name space was modified while the iteration was in progress.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODISOStr` type (page 845).

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more entries remain in the name space. This method returns `kODTrue` if the preceding call to the `First` or `Next` method found an entry. This method returns `kODFalse` when you have examined all the entries. If the name space is empty, this method always returns `kODFalse`.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The name space was modified while the iteration was in progress.

Next

The `Next` method obtains the next entry in the name space.

```
void Next (out ODISOStr key,
           out ODByteArray value);
```

`key` A pointer to an ISO string representing the key in the next entry in the name space, or `kODNULL` if you have reached the end of the name space.

`value` A byte array whose buffer contains the value for the next entry in the name space.

DISCUSSION

If your part calls this method before calling this value iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

It is your responsibility to deallocate the ISO string and byte array structure (and its buffer) when they are no longer needed. You must also delete the key when it is no longer needed.

EXCEPTIONS

`kODErrIteratorOutOfSync` The name space was modified while the iteration was in progress.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODISOStr` type (page 845).

ODValueNameSpace

Superclasses `ODNameSpace` → `ODObject`

Subclasses none

An object of the `ODValueNameSpace` class is a collection of values, each of which has a unique key to identify the value within the collection.

Description

A value name space allows a part to identify a value using a unique key, which can be passed easily between parts. A part can create a value name space to store information about other part editors, such as the part's supported part kinds. The values in a value name space are stored as byte arrays whose buffers can contain values of any types.

Your part can create objects of the `ODValueNameSpace` class by calling the name-space manager's `CreateNameSpace` method (page 422), passing the constant `kODNSDataTypeODValue` for the type of the name space.

Value name spaces can be arranged hierarchically to allow you to search multiple value name spaces for a single key. Searches move from a child value name space to its parent value name space until the entry is found or until there are no more value name spaces to search.

Methods

This section presents summary descriptions of the `ODValueNameSpace` methods, followed by detailed descriptions in alphabetical order.

`CreateIterator`	Creates a value iterator for the entries in this value name space.
`GetEntry`	Searches for an entry with the specified key and, if it exists, gets the value associated with that key.

`Register`	Adds a new entry to this value name space.

CreateIterator

The `CreateIterator` method creates a value iterator for the entries in this value name space.

```
ODValueIterator CreateIterator ();
```

return value A reference to a value iterator for iterating over the entries in this value name space.

DISCUSSION

You call this method when you need to apply an operation to all the entries of this name space.

While you are using the returned value iterator, you must not modify this value name space; in particular, you must not register or unregister entries and you must not delete this value name space.

You must delete the returned value iterator when it is no longer needed.

SEE ALSO

The `ODValueIterator` class (page 789).

GetEntry

The `GetEntry` method searches for an entry with the specified key and, if it exists, gets the value associated with that key.

```
ODBoolean GetEntry (in ODISOStr key,
                    out ODByteArray value);
```

`key`	The key to search for in this value name space.
`value`	A byte array structure to contain the value corresponding to the specified key, if the entry is found.
return value	`kODTrue` if the entry was found, otherwise `kODFalse`.

DISCUSSION

If the specified key is found, this method copies the entry's associated value into the `value` parameter. If the specified key is not found in this name space, this method searches the parent name space. Searches proceed from each value name space to its parent until one of the following happens: the entry is found, there is no parent name space to search, or the parent name space is an object name space instead of a value name space.

If the key is found, the `_buffer` field of the `value` output parameter is set to point to a memory block containing the a copy of the value associated with the key. If the key is not found after searching this value name space and its ancestors, the `_buffer` field of the `value` parameter is set to `kODNULL`.

Your part must delete the byte array and its buffer when they are no longer needed.

SEE ALSO

The `ODByteArray` type (page 847).
The `ODISOStr` type (page 845).
The `ODNameSpace::Exists` method (page 415).
The `ODValueNameSpace::Register` method (page 795).

Register

The `Register` method adds a new entry to this value name space.

```
void Register (in ODISOStr key,
               in ODByteArray value);
```

`key`	The key for the new entry.

`value` A byte array whose buffer contains the value to associate with the key.

DISCUSSION

This method copies both the `key` and the `value` parameter's buffer into a new entry in the name space. If the specified key already exists within this name space, this method overwrites the old value with the new one and deallocates the memory used by the old value.

SEE ALSO

The `ODByteArray` type (page 847)
The `ODISOStr` type (page 845)
The `ODNameSpace::Unregister` method (page 419).

ODWindow

Superclasses `ODRefCntObject` → `ODObject`

Subclasses none

An object of the `ODWindow` class is a wrapper for a platform-specific window structure.

Description

Every window (except a modal dialog box) must be associated with an OpenDoc window object so that the part belonging to the root frame of the window, and its embedded parts, can receive events from the dispatcher. To make platform-specific calls, part editors can retrieve the platform-specific window from the window object. However, in most cases, the interface to the `ODWindow` class provides the capability you need for interacting with your platform-specific windows.

Your part creates a window object by calling the window-state object's `RegisterWindow` method (page 835) or `RegisterWindowForFrame` method (page 837).

When your part creates a window, it specifies whether the new window should be a root window. A document remains open as long as it has an open root window; the document shell closes the document when the last root window is closed. A root window is also called a document window. The initial window of a document is a root window; its root part is the root part of the document. Part windows, which display embedded parts, and dialog boxes are not root windows. OpenDoc permits a single document to have multiple root windows as long as the root part provides a user interface to support them.

Your part should not maintain references to window objects for accessing OpenDoc windows because the document shell or the window-state object can close the window object and invalidate the reference. You should instead maintain window IDs, from which the window objects can be reconstructed.

Methods

This section presents summary descriptions of the `ODWindow` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are called only by the document shell or container applications. Methods marked [M] are specific to the Mac OS platform.

Window Manipulation

`Close` [D]	Closes this window object.
`CloseAndRemove`	Closes this window object and removes the root frame from its draft.
`Hide`	Makes this window invisible.
`Show`	Makes this window visible.
`Select` [M]	Makes this window object the frontmost window and activates it.
`GetID`	Returns the window ID for this window object.
`GetPlatformWindow`	Returns the platform-specific window for this window object.
`Update` [M]	Forces immediate updating of this window, rather than waiting for an update event.

Window Characteristics

`IsActive`	Returns a Boolean value that indicates whether this window is active.
`IsFloating`	Returns a Boolean value that indicates whether this window is a floating window.
`IsResizable`	Returns a Boolean value that indicates whether this window is resizable.
`IsRootWindow`	Returns a Boolean value that indicates whether this window object is a root window.
`IsShown`	Returns a Boolean value that indicates whether this window is currently visible.
`ShouldDispose` [M]	Returns a Boolean value that indicates whether the platform window should be disposed of when this window object is deleted.

`ShouldSave`	Returns a Boolean value that indicates whether this window object should be saved in its draft.
`SetShouldSave`	Specifies whether this window object should be saved in its draft.
`ShouldShowLinks`	Returns a Boolean value that indicates whether links should be highlighted in this window.
`SetShouldShowLinks`	Specifies whether links should be highlighted in this window.

Facet Manipulation

`GetFacetUnderPoint`	Returns a reference to the facet of this window under the specified point.
`GetRootFacet`	Returns a reference to the root facet of this window object.
`Open`	Creates a facet hierarchy in this window object.

Frame Manipulation

`GetRootFrame`	Returns a reference to the root frame of this window object.
`AcquireSourceFrame`	Returns a reference to the frame, if any, from which this window was opened.
`SetSourceFrame`	Sets the source frame of this window object.
`AdjustWindowShape`	Reshapes the root frame to match the size of this window.

AcquireSourceFrame

The `AcquireSourceFrame` method returns a reference to the frame, if any, from which this window was opened.

```
ODFrame AcquireSourceFrame ();
```

return value A reference to the frame from which this window was opened, or `kODNULL` if the window does not have a source frame (for example, a root window).

DISCUSSION

This method increments the reference count of the returned frame. When you have finished using that frame, you should call its `Release` method.

If your part class implements shared dialog windows (which are shared by all part objects of your class in a document), your part objects can use the source frame of the dialog window to find out which frame is currently associated with the dialog window.

It is possible to change the source frame later by calling this window's `SetSourceFrame` method.

SEE ALSO

The `ODWindow::SetSourceFrame` method (page 808).

AdjustWindowShape

The `AdjustWindowShape` method reshapes the root frame to match the size of this window.

```
void AdjustWindowShape ();
```

DISCUSSION

Your part (the root part) calls this method after resizing the window.

Close

Document shell

The `Close` method closes this window object.

```
void Close ();
```

DISCUSSION

OpenDoc calls this method when closing a document; parts typically call this window object's `CloseAndRemove` method instead of this method.

CloseAndRemove

The `CloseAndRemove` method closes this window object and removes the root frame from its draft.

```
void CloseAndRemove ();
```

DISCUSSION

This method closes this window object, releases this window object, deletes the root facet, and removes the root frame from the draft. Your part calls this method to remove auxiliary windows such as palettes or part windows.

GetFacetUnderPoint

The `GetFacetUnderPoint` method returns a reference to the facet of this window under the specified point.

```
ODFacet GetFacetUnderPoint (in ODPoint aPoint);
```

`aPoint`	The location to test, expressed in window coordinates.
return value	A reference to the facet under the specified point.

DISCUSSION

OpenDoc calls this method for event dispatching; this method is not called by most parts.

If several nested facets are under the point, the innermost one is returned. If multiple overlapping frames are under the point, the unobscured (that is, the

frontmost) one is returned. The bundled and selected properties of frames and facets are respected.

SEE ALSO

The `ODPoint` type (page 855).

GetID

The `GetID` method returns the window ID for this window object.

```
ODID GetID ();
```

return value The window ID for this window object.

DISCUSSION

The window-state object assigns window IDs that are valid for the length of the session. You can use this method to get the window ID of this window object when it is created, and then pass that ID to the window-state object's `AcquireWindow` method for subsequent access to this window object.

SEE ALSO

The `ODID` type (page 869).
The `ODWindowState::AcquireWindow` method (page 824).

GetPlatformWindow

The `GetPlatformWindow` method returns the platform-specific window for this window object.

```
ODPlatformWindow GetPlatformWindow ();
```

return value A 32-bit value identifying the platform-specific window for this window object. On the Mac OS platform, the return value is a window pointer (type `WindowPtr`).

GetRootFacet

The `GetRootFacet` method returns a reference to the root facet of this window object.

```
ODFacet GetRootFacet ();
```

return value A reference to the root facet of this window object.

GetRootFrame

The `GetRootFrame` method returns a reference to the root frame of this window object.

```
ODFrame GetRootFrame ();
```

return value A reference to the root frame of this window object.

DISCUSSION

This method does not increment the reference count of the returned frame. For that reason, if you cache the returned frame, you should call its `Acquire` method to increment its reference count and then call its `Release` method when you have finished using it.

Hide

The `Hide` method makes this window invisible.

```
void Hide ();
```

SEE ALSO

The `ODWindow::Show` method (page 809).

IsActive

The `IsActive` method returns a Boolean value that indicates whether this window is active.

```
ODBoolean IsActive ();
```

return value `kODTrue` if this window is active, otherwise `kODFalse`.

DISCUSSION

A window is active if it is either the frontmost nonfloating window or a floating window.

IsFloating

The `IsFloating` method returns a Boolean value that indicates whether this window is a floating window.

```
ODBoolean IsFloating ();
```

return value `kODTrue` if this window is a floating window, otherwise `kODFalse`.

IsResizable

The `IsResizable` method returns a Boolean value that indicates whether this window is resizable.

```
ODBoolean IsResizable ();
```

return value `kODTrue` if this window is resizable, otherwise `kODFalse`.

IsRootWindow

The `IsRootWindow` method returns a Boolean value that indicates whether this window object is a root window.

```
ODBoolean IsRootWindow ();
```

return value `kODTrue` if this window object is a root window, otherwise `kODFalse`.

IsShown

The `IsShown` method returns a Boolean value that indicates whether this window is currently visible.

```
ODBoolean IsShown ();
```

return value `kODTrue` if this window is currently visible, otherwise `kODFalse`.

SEE ALSO

The `ODWindow::Hide` method (page 804).
The `ODWindow::Show` method (page 809).

Open

The `Open` method creates a facet hierarchy in this window object.

```
void Open ();
```

DISCUSSION

Your part calls this method when first opening a window. This method does not make the window visible or change window ordering; for those operations, your part calls this window object's `Show` and `Select` methods instead.

SEE ALSO

The `ODWindow::Select` method (page 806).
The `ODWindow::Show` method (page 809).

Select

Mac OS

The `Select` method makes this window the frontmost window and activates it.

```
void Select ();
```

DISCUSSION

Your part calls this method when first opening or when later activating a window.

This method changes window ordering.

SEE ALSO

The `ODWindow::Show` method (page 809).

SetShouldSave

The `SetShouldSave` method specifies whether this window object should be saved in its draft.

```
void SetShouldSave (in ODBoolean shouldSave);
```

`shouldSave` `kODTrue` if this window object should be saved in its draft, otherwise `kODFalse`.

DISCUSSION

This property should generally be set to true for root windows and false for other windows.

SEE ALSO

The `ODWindow::ShouldSave` method (page 808).

SetShouldShowLinks

The `SetShouldShowLinks` method specifies whether links should be highlighted in this window.

```
void SetShouldShowLinks (in ODBoolean shouldShowLinks);
```

`shouldShowLinks`
`kODTrue` if links should be highlighted in this window, otherwise `kODFalse`.

SEE ALSO

The `ODWindow::ShouldShowLinks` method (page 809).

SetSourceFrame

The `SetSourceFrame` method sets the source frame of this window object.

```
void SetSourceFrame (in ODFrame frame);
```

`frame` A reference to the new source frame.

SEE ALSO

The `ODWindow::AcquireSourceFrame` method (page 799).

ShouldDispose

Mac OS

The `ShouldDispose` method returns a Boolean value that indicates whether the platform window should be disposed of when this window object is deleted.

```
ODBoolean ShouldDispose ();
```

return value `kODTrue` if the platform window should be disposed of when this window object is deleted, otherwise `kODFalse`.

ShouldSave

The `ShouldSave` method returns a Boolean value that indicates whether this window object should be saved in its draft.

```
ODBoolean ShouldSave ();
```

return value `kODTrue` if this window object should be saved in its draft, otherwise `kODFalse`.

SEE ALSO

The `ODWindow::SetShouldSave` method (page 807).

ShouldShowLinks

The `ShouldShowLinks` method returns a Boolean value that indicates whether links should be highlighted in this window.

```
ODBoolean ShouldShowLinks ();
```

return value `kODTrue` if links should be highlighted in this window, otherwise `kODFalse`.

SEE ALSO

The `ODWindow::SetShouldShowLinks` method (page 807).

Show

The `Show` method makes this window visible.

```
void Show ();
```

DISCUSSION

Your part calls this method when first opening a window and when making a window visible after having called its `Hide` method.

This method does not change window ordering.

SEE ALSO

The `ODWindow::Hide` method (page 804).
The `ODWindow::Select` method (page 806).

Update

Mac OS

The `Update` method forces immediate updating of this window, rather than waiting for an update event.

```
void Update ();
```

DISCUSSION

When an update event occurs that involves a facet of your part, OpenDoc calls this method, which in turn calls its facet's `Update` method, which then calls the `Draw` method associated with the facet's part. Your part might also call this method to force updating when it does not happen automatically, for instance, when there is a mouse-down event and you cannot wait for an update event.

SEE ALSO

The `ODFacet::Update` method (page 251).
The `ODPart::Draw` method (page 487).

ODWindowIterator

Superclasses `ODObject`

Subclasses none

An object of the `ODWindowIterator` class provides access to all windows of the window-state object.

Description

You use a window iterator to apply an operation to all windows of all open drafts of the current session's document. For example, a root part might use a window iterator to tile all the currently open windows. A window iterator maintains a reference to its window-state object and to the current window object. The internal list of windows in the window-state object is ordered by creation time and is not related to front-to-back ordering of the windows.

Your part creates a window iterator object by calling the window-state object's `CreateWindowIterator` method (page 830), which returns a reference to a window iterator object.

While you are using a window iterator, you should not modify the list of open windows. You must postpone adding windows to or removing windows from the list of open windows until after you have deleted the iterator.

For more information related to the window-state object, see the `ODWindowState` class description (page 817). For more information on accessing objects through iterators, see the chapter on OpenDoc runtime features in the *OpenDoc Programmer's Guide for the Mac OS*.

Methods

This section presents summary descriptions of the `ODWindowIterator` methods grouped according to purpose, followed by detailed descriptions in alphabetical order.

Accessing

`First`	Begins the iteration and returns a reference to the first window in the window state.
`Last`	Begins the iteration and returns a reference to the last window in the window state.
`Next`	Returns a reference to the next window in the window state.
`Previous`	Returns a reference to the previous window in the window state.

Iterator Testing

`IsNotComplete`	Returns a Boolean value that indicates whether the iteration is incomplete.

First

The `First` method begins the iteration and returns a reference to the first window in the window state.

```
ODWindow First ();
```

return value A reference to the first window in the window state.

DISCUSSION

If you are iterating from the first window to the last, your part must call this method before calling this window iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

Call the `Next` method to step through the window list from first to last.

This method does not increment the reference count of the returned window object.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of open windows was modified while the iteration was in progress.

IsNotComplete

The `IsNotComplete` method returns a Boolean value that indicates whether the iteration is incomplete.

```
ODBoolean IsNotComplete ();
```

return value `kODTrue` if the iteration is incomplete, otherwise `kODFalse`.

DISCUSSION

Your part calls this method to test whether more windows remain in the window state. This method returns `kODTrue` if the preceding call to the `First`, `Last`, `Next`, or `Previous` method found a window. This method returns `kODFalse` when you have examined all the windows.

EXCEPTIONS

`kODErrIteratorNotInitialized`
: This method was called before calling either the `First` or `Next` method to begin the iteration.

`kODErrIteratorOutOfSync`
: The list of open windows was modified while the iteration was in progress.

Last

The `Last` method begins the iteration and returns a reference to the last window in the window state.

```
ODWindow Last ();
```

return value A reference to the last window in the window state.

DISCUSSION

If you are iterating from the last window to the first, your part must call this method before calling this window iterator's `IsNotComplete` method for the first time. This method may be called multiple times; each time resets the iteration.

Call the `Previous` method to step through the window list from last to first.

This method does not increment the reference count of the returned window object.

EXCEPTIONS

`kODErrIteratorOutOfSync`
The list of open windows was modified while the iteration was in progress.

Next

The `Next` method returns a reference to the next window in the window state.

```
ODWindow Next ();
```

return value A reference to the next window in the window state, or `kODNULL` if you have reached the last window.

DISCUSSION

If your part calls this method before calling this window iterator's `First` method to begin the iteration, then this method works the same as calling the `First` method.

This method does not increment the reference count of the returned window object.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of open windows was modified while the iteration was in progress.

Previous

The `Previous` method returns a reference to the previous window in the window state.

```
ODWindow Previous ();
```

return value A reference to the previous window in the window state, or `kODNULL` if you have reached the first window.

DISCUSSION

If your part calls this method before calling this window iterator's `Last` method to begin the iteration, then this method works the same as calling the `Last` method.

This method does not increment the reference count of the returned window object.

EXCEPTIONS

`kODErrIteratorOutOfSync`
: The list of open windows was modified while the iteration was in progress.

ODWindowState

Superclasses `ODObject`

Subclasses none

An object of the `ODWindowState` class maintains a list of all the open window objects for all open drafts in an OpenDoc session and references to the base menu bar and the current menu bar.

Description

When a document is opened, the session object creates a single window-state object. All parts of that document share the window-state object; you can obtain a reference to it by calling the session object's `GetWindowState` method (page 589). The document shell and dispatcher use the window-state object to pass events to parts so the parts can activate themselves, handle user input, and adjust their menus as necessary. The document shell may manage more than one open document. Typically, however, there is only one open document in a session, but multiple drafts of the document may be open. A part may be displayed in any number of frames in any window of a document. The dispatcher passes events to the correct part, no matter what window encloses the active frame and how many other frames the part has.

Your part accesses the window-state object to create new windows, to access a particular window, and to access the base menu bar object.

For more information related to window objects, see the `ODWindow` class description (page 797).

Methods

This section presents summary descriptions of the `ODWindowState` methods grouped according to purpose, followed by detailed descriptions in alphabetical order. Methods marked [D] are called only by the document shell

or container applications. Methods marked [M] are specific to the Mac OS platform.

Window Creation

`RegisterWindow` Creates an OpenDoc window object and root frame for the specified platform-specific window.

`RegisterWindowForFrame` Creates a window object for the specified platform-specific window and root frame.

Window Manipulation

`ActivateFrontWindows` [M] Activates the frontmost root window and all the floating windows.

`DeactivateFrontWindows` [M] Deactivates the frontmost nonfloating window and all the floating windows.

`CloseWindows` [D] Closes all windows belonging to the specified draft.

`OpenWindows` [D] Opens all windows belonging to the specified draft.

`AcquireWindow` Returns a reference to the window object with the specified ID.

`AcquireODWindow` Returns a reference to the window object corresponding to the specified window structure.

`AcquireActiveWindow` Returns a reference to the frontmost nonfloating window.

`AcquireFrontWindow` [M] Returns a reference to the frontmost window.

`AcquireFrontFloatingWindow` [M] Returns a reference to the frontmost floating window.

`AcquireFrontRootWindow` [M] Returns a reference to the frontmost (nonfloating) root window.

`Externalize` [D] Writes to persistent storage the window properties of windows of the specified draft and saves their frames as the root frames of the draft.

`Internalize` [D] Reads into memory all root frames of the specified draft, causing their parts to open the windows.

`CreateWindowIterator`
Creates a window iterator object for the windows (of all drafts) in this window-state object.

`SetDefaultWindowTitles` [D]
Synchronizes window titles with the filename of the document of the specified draft.

Window Characteristics

`IsODWindow` Returns a Boolean value that indicates whether this window-state object has a window with the specified platform-specific window.

`GetWindowCount` Returns the number of windows (of all open drafts) in this window-state object.

`GetRootWindowCount` Returns the number of root windows belonging to the specified draft.

`GetTotalRootWindowCount`
Returns the total number of root windows of all open drafts.

Menu Bar Manipulation

`CreateMenuBar` [D] Creates and initializes a menu bar object.

`AcquireCurrentMenuBar` [D] [M]
Returns a reference to the current menu bar object.

`AcquireBaseMenuBar` [D] [M]
Returns a reference to the base menu bar object.

`CopyBaseMenuBar` Copies the base menu bar object.

`SetBaseMenuBar` [D] Installs the specified menu bar object as the base menu for parts to copy.

`AdjustPartMenus` Prepares both the root part and the part with the menu focus to display their menus.

Facet and Canvas Creation

`CreateFacet` Creates a facet object.

`CreateCanvas` Creates a canvas object.

AcquireActiveWindow

The `AcquireActiveWindow` method returns a reference to the frontmost nonfloating window.

```
ODWindow AcquireActiveWindow ();
```

return value A reference to the frontmost nonfloating window, or `kODNULL` if the frontmost nonfloating window is not an OpenDoc window.

DISCUSSION

This method increments the reference count of the returned window. When you have finished using that window, you should call its `Release` method.

AcquireBaseMenuBar

Document shell
Mac OS

The `AcquireBaseMenuBar` method returns a reference to the base menu bar object.

```
ODMenuBar AcquireBaseMenuBar ();
```

return value A reference to the base menu bar object.

DISCUSSION

The document shell calls this method. Your part typically calls this window-state object's `CopyBaseMenuBar` method instead of this method.

This method increments the reference count of the returned menu bar. When the caller has finished using that menu bar, it should call the menu bar's `Release` method.

SEE ALSO

The `ODWindowState::CopyBaseMenuBar` method (page 826).
The `ODWindowState::SetBaseMenuBar` method (page 839).

AcquireCurrentMenuBar

Document shell
Mac OS

The `AcquireCurrentMenuBar` method returns a reference to the current menu bar object.

```
ODMenuBar AcquireCurrentMenuBar ();
```

return value A reference to the current menu bar object.

DISCUSSION

The document shell calls this method. Your part typically calls this window-state object's `CopyBaseMenuBar` method instead of this method.

This method increments the reference count of the returned menu bar. When the caller has finished using that menu bar, it should call the menu bar's `Release` method.

SEE ALSO

The `ODWindowState::CopyBaseMenuBar` method (page 826).

AcquireFrontFloatingWindow

Mac OS

The `AcquireFrontFloatingWindow` method returns a reference to the frontmost floating window.

```
ODWindow AcquireFrontFloatingWindow ();
```

return value A reference to the frontmost floating window, or `kODNULL` if there are no floating OpenDoc windows.

DISCUSSION

This method increments the reference count of the returned window. When you have finished using that window, you should call its `Release` method.

AcquireFrontRootWindow

Mac OS

The `AcquireFrontRootWindow` method returns a reference to the frontmost (nonfloating) root window.

```
ODWindow AcquireFrontRootWindow ();
```

return value A reference to the frontmost (nonfloating) root window.

DISCUSSION

This method increments the reference count of the returned window. When you have finished using that window, you should call its `Release` method.

AcquireFrontWindow

Mac OS

The `AcquireFrontWindow` method returns a reference to the frontmost window.

```
ODWindow AcquireFrontWindow ();
```

return value A reference to the frontmost window, or `kODNULL` if the frontmost window is not an OpenDoc window.

DISCUSSION

This method increments the reference count of the returned window. When you have finished using that window, you should call its `Release` method.

AcquireODWindow

The `AcquireODWindow` method returns a reference to the window object corresponding to the specified platform-specific window.

```
ODWindow AcquireODWindow (in ODPlatformWindow aWindow);
```

`aWindow` A 32-bit value identifying a platform-specific window. On the Mac OS platform, this parameter is a window pointer (type `WindowPtr`).

return value A reference to the window object corresponding to the specified platform-specific window, or `kODNULL` if the window is not an OpenDoc window.

DISCUSSION

OpenDoc calls this method.

This method increments the reference count of the returned window. When the caller has finished using that window, it should call the window's `Release` method.

SEE ALSO

The `ODWindowState::IsODWindow` method (page 834).

AcquireWindow

The `AcquireWindow` method returns a reference to the window object with the specified ID.

```
ODWindow AcquireWindow (in ODID id);
```

`id`	The window ID for the window object.
return value	A reference to the window object with the specified ID, or `kODNULL` if the window has been deleted or does not exist.

DISCUSSION

This method increments the reference count of the returned window. When you have finished using that window, you should call its `Release` method.

SEE ALSO

The `ODID` type (page 869).
The `ODWindow::GetID` method (page 802).

ActivateFrontWindows

Mac OS

The `ActivateFrontWindows` method activates the frontmost root window and all the floating windows.

```
void ActivateFrontWindows ();
```

DISCUSSION

Your part calls this method after a modal dialog box is dismissed.

SEE ALSO

The `ODWindowState::DeactivateFrontWindows` method (page 830).

AdjustPartMenus

The `AdjustPartMenus` method prepares both the root part and the part with the menu focus to display their menus.

```
void AdjustPartMenus ();
```

DISCUSSION

OpenDoc calls this method when a mouse-down event occurs in the menu bar. This method in turn calls the `AdjustMenus` method for both the root part and the part with the menu focus, so the parts can enable or disable menu items as necessary. If the root part has the menu focus, then OpenDoc only calls the `AdjustMenus` method once.

SEE ALSO

The `ODPart::AdjustMenus` method (page 465).

CloseWindows

Document shell

The `CloseWindows` method closes all windows belonging to the specified draft.

```
void CloseWindows (in ODDraft draft);
```

`draft` A reference to the open draft object.

DISCUSSION

The document shell calls this method when closing a draft.

SEE ALSO

The `ODWindowState::OpenWindows` method (page 834).

CopyBaseMenuBar

The `CopyBaseMenuBar` method copies the base menu bar object.

```
ODMenuBar CopyBaseMenuBar ();
```

return value A reference to the newly created copy of the base menu bar object.

DISCUSSION

Your part calls this method to create a menu bar object to which it can add its own menus.

This method initializes the reference count of the returned menu bar. When you have finished using that menu bar, you should call its `Release` method.

SEE ALSO

The `ODMenuBar::Copy` method (page 388).
The `ODWindowState::AcquireBaseMenuBar` method (page 820).
The `ODWindowState::SetBaseMenuBar` method (page 839).

CreateCanvas

The `CreateCanvas` method creates a canvas object.

```
ODCanvas CreateCanvas (in ODGraphicsSystem graphicsSystem,
                       in ODPlatformCanvas platformCanvas,
                       in ODBoolean isDynamic,
                       in ODBoolean isOffscreen);
```

`graphicsSystem`
A 16-bit value specifying the graphics system you want to use for this canvas. Valid graphics systems are platform dependent.

`platformCanvas`
A 32-bit value identifying the graphics-system-specific drawing structure to assign to the canvas, or `kODNULL` for no drawing structure. Valid values for `platformCanvas` are graphics-system-dependent.

`isDynamic` `kODTrue` if the canvas is to be dynamic, otherwise `kODFalse`.

`isOffscreen`
`kODTrue` if the canvas is to be offscreen, otherwise `kODFalse`.

return value A reference to the newly created canvas object.

DISCUSSION

Your part calls this method to create a canvas object that will not be attached to any facet. To create a canvas to attach to a particular facet, you should call that facet's `CreateCanvas` method instead of this method.

On the Mac OS platform, the graphics system may be either QuickDraw (`kODQuickDraw`) or QuickDraw GX (`kODQuickDrawGX`). For QuickDraw, the platform canvas should be a QuickDraw graphics port (type `GrafPtr`); for

QuickDraw GX, it should be a QuickDraw GX view port object (type `gxViewPort`).

SEE ALSO

The `ODGraphicsSystem` type (page 853).
The `ODFacet::CreateCanvas` method (page 232).
The `ODCanvas` class (page 61).

CreateFacet

The `CreateFacet` method creates a facet object.

```
ODFacet CreateFacet (in ODFrame frame,
                     in ODShape clipShape,
                     in ODTransform externalTransform,
                     in ODCanvas canvas,
                     in ODCanvas biasCanvas);
```

`frame` A reference to the frame for the facet.

`clipShape` A reference to the initial clip shape for the facet.

`externalTransform`
A reference to the initial external transform for the facet.

`canvas` A reference to the canvas the facet should draw to, or `kODNULL` if identical to the canvas associated with the containing facet.

`biasCanvas` A reference to the canvas object to whose coordinate space the geometry is biased, or `kODNULL` if the geometry is in the standard platform-normal coordinate space.

return value A reference to the newly created facet object.

DISCUSSION

Your part calls this method to create a root facet (for example, for printing). The frame is defined for the lifetime of the facet object; once set, it cannot be changed.

To create a facet object for a visible embedded frame, your part should call its own display facet's `CreateEmbeddedFacet` method instead of this method.

SEE ALSO

The `ODFacet::CreateEmbeddedFacet` method (page 234).
The `ODFacet` class (page 215).

CreateMenuBar

Document shell

The `CreateMenuBar` method creates and initializes a menu bar object.

```
ODMenuBar CreateMenuBar (in ODPlatformMenuBar menuBar);
```

`menuBar`	A 32-bit value identifying a menu bar. On the Mac OS platform, this parameter is a handle (type `Handle`).
return value	A reference to the newly created menu bar object.

DISCUSSION

The document shell calls this method. Your part typically calls this window-state object's `CopyBaseMenuBar` method instead of this method.

This method initializes the reference count of the returned menu bar. When the caller has finished using that menu bar, it should call the menu bar's `Release` method.

SEE ALSO

The `ODWindowState::CopyBaseMenuBar` method (page 826).
The `ODMenuBar` class (page 383).

CreateWindowIterator

The `CreateWindowIterator` method creates a window iterator for the windows (of all drafts) in this window-state object.

```
ODWindowIterator CreateWindowIterator ();
```

return value A reference to the newly created window iterator.

DISCUSSION

Your part calls this method if it needs to apply an operation to all windows of this window-state object. For example, a root part might use a window iterator to tile all the currently open windows.

While you are using the returned window iterator, you must not call any methods that create or delete windows.

You must delete the returned window iterator when it is no longer needed.

SEE ALSO

The `ODWindowIterator` class (page 811).

DeactivateFrontWindows

Mac OS

The `DeactivateFrontWindows` method deactivates the frontmost nonfloating window and all the floating windows.

```
void DeactivateFrontWindows ();
```

DISCUSSION

Your part calls this method before displaying a modal dialog box.

SEE ALSO

The `ODWindowState::ActivateFrontWindows` method (page 825).

Externalize

Document shell

The `Externalize` method writes to persistent storage the window properties of windows of the specified draft and saves their frames as the root frames of the draft.

```
void Externalize (in ODDraft draft);
```

`draft` A reference to the open draft object.

DISCUSSION

The document shell calls this method when saving a draft. This method saves window properties for those windows of the specified draft that should be saved, that is, for those windows whose `ShouldSave` method returns true.

SEE ALSO

The `ODWindow::ShouldSave` method (page 808).
The `ODWindowState::Internalize` method (page 833).

GetRootWindowCount

The `GetRootWindowCount` method returns the number of root windows belonging to the specified draft.

```
ODUShort GetRootWindowCount (in ODDraft draft);
```

`draft` A reference to the open draft object.

return value The number of root windows belonging to the specified draft, expressed as an unsigned 16-bit value.

DISCUSSION

The document shell calls this method when closing a window to determine whether to close the draft; a draft is closed when its last root window is closed.

SEE ALSO

The `ODWindowState::GetTotalRootWindowCount` method (page 832).

GetTotalRootWindowCount

The `GetTotalRootWindowCount` method returns the total number of root windows of all open drafts.

```
ODUShort GetTotalRootWindowCount ();
```

return value The number of root windows of all drafts, expressed as an unsigned 16-bit value.

SEE ALSO

The `ODWindowState::GetRootWindowCount` method (page 831).

GetWindowCount

The `GetWindowCount` method returns the number of windows (of all open drafts) in this window-state object.

```
ODUShort GetWindowCount ();
```

return value The total number of windows in this window-state object, expressed as an unsigned 16-bit value.

Internalize

Document shell

The `Internalize` method reads into memory all root frames of the specified draft, causing their parts to open the windows.

```
void Internalize (in ODDraft draft);
```

`draft` A reference to the open draft object.

DISCUSSION

The document shell calls this method when opening a draft. After reading the root frames, this method reads into memory the part associated with each root frame, then passes that root frame as the parameter to its part's `Open` method. The `Open` method, in turn, creates the root (document) window for the root frame.

EXCEPTIONS

`kODErrOutOfMemory` There is not enough memory to read in the data.

SEE ALSO

The `ODPart::Open` method (page 513).
The `ODWindowState::Externalize` method (page 831).

IsODWindow

The `IsODWindow` method returns a Boolean value that indicates whether this window-state object has a window with the specified platform-specific window.

```
ODBoolean IsODWindow (in ODPlatformWindow aWindow);
```

`aWindow`	A 32-bit value identifying a platform-specific window. On the Mac OS platform, this parameter is a window pointer (type `WindowPtr`).
return value	`kODTrue` if this window-state object has a window with the specified platform-specific window, otherwise `kODFalse`.

DISCUSSION

OpenDoc calls this method.

SEE ALSO

The `ODWindowState::AcquireODWindow` method (page 823).

OpenWindows

Document shell

The `OpenWindows` method opens all windows belonging to the specified draft.

```
void OpenWindows (in ODDraft draft);
```

`draft`	A reference to the open draft object.

DISCUSSION

If the draft is already open, then this method brings its windows to the front.

EXCEPTIONS

`kODErrOutOfMemory`	There is not enough memory to open all the windows.

SEE ALSO

The `ODWindowState::CloseWindows` method (page 826).

RegisterWindow

The `RegisterWindow` method creates an OpenDoc window object and root frame for the specified platform-specific window.

```
ODWindow RegisterWindow (in ODPlatformWindow newWindow,
                         in ODType frameType,
                         in ODBoolean isRootWindow,
                         in ODBoolean isResizable,
                         in ODBoolean isFloating,
                         in ODBoolean shouldSave,
                         in ODBoolean shouldDispose,
                         in ODPart rootPart,
                         in ODTypeToken viewType,
                         in ODTypeToken presentation,
                         in ODFrame sourceFrame);
```

`newWindow` A 32-bit value identifying a platform-specific window. On the Mac OS platform, this parameter is a window pointer (type `WindowPtr`).

`frameType` The type of root frame for the window object. The frame type must be either regular frame (`kODFrameObject`) or nonpersistent frame (`kODNonPersistentFrameObject`).

`isRootWindow`
`kODTrue` if the window object is to be a root window, otherwise `kODFalse`.

`isResizable`
: `kODTrue` if the window object is to be resizable, otherwise `kODFalse`.

`isFloating` `kODTrue` if the window object is to be a floating window, otherwise `kODFalse`.

`shouldSave` `kODTrue` if the window object is to be saved in its draft, otherwise `kODFalse`.

`shouldDispose`
: `kODTrue` if the platform window should be disposed of when the window object is deleted, otherwise `kODFalse`.

`rootPart` A reference to the part associated with the root frame of the window.

`viewType` A tokenized string representing the view type for the root frame of the window.

`presentation`
: A tokenized string representing the presentation type for the root frame of the window.

`sourceFrame`
: A reference to the frame from which the window object was opened (used when an embedded frame is opened into a window), or `kODNULL` if the frame does not exist.

return value A reference to the newly created window object.

DISCUSSION

Your part calls this method to create a window object when there is no pre-existing root frame. You should create the platform-specific window as invisible. After calling this method, you should call the `Show` method of the returned window to make it visible.

The `viewType` parameter must be the tokenized form of one of the view-type constants (`kODViewAsFrame`, `kODViewAsLargeIcon`, `kODViewAsSmallIcon`, or `kODViewAsThumbnail`). You can call the session object's `Tokenize` method to obtain a token corresponding to one of these constants.

This method initializes the reference count of the returned window. When you have finished using that window, you should call its `Release` method.

EXCEPTIONS

`kODErrCannotCreateWindow`
: There is not enough memory to create the new window object.

SEE ALSO

The `ODType` type (page 846).
The `ODTypeToken` type (page 847).
The `ODSession::Tokenize` method (page 598).
The `ODWindow::Show` method (page 809).
The `ODWindowState::RegisterWindowForFrame` method (page 837).

RegisterWindowForFrame

The `RegisterWindowForFrame` method creates a window object for the specified platform-specific window and root frame.

```
ODWindow RegisterWindowForFrame (
                            in ODPlatformWindow newWindow,
                            in ODFrame frame,
                            in ODBoolean isRootWindow,
                            in ODBoolean isResizable,
                            in ODBoolean isFloating,
                            in ODBoolean shouldSave,
                            in ODBoolean shouldDispose,
                            in ODFrame sourceFrame);
```

`newWindow` A 32-bit value identifying a platform-specific window. On the Mac OS platform, this parameter is a window pointer (type `WindowPtr`).

`frame` A reference to the root frame of the window object.

`isRootWindow`
: `kODTrue` if the window object is to be a root window, otherwise `kODFalse`.

`isResizable`
: `kODTrue` if the window object is to be resizable, otherwise `kODFalse`.

`isFloating`
: `kODTrue` if the window object is to be a floating window, otherwise `kODFalse`.

`shouldSave`
: `kODTrue` if the window object is to be saved in its draft, otherwise `kODFalse`.

`shouldDispose`
: `kODTrue` if the platform window should be disposed of when the window object is deleted, otherwise `kODFalse`.

`sourceFrame`
: A reference to the frame from which the window object was opened (used when an embedded frame is opened into a window), or `kODNULL` if the frame does not exist.

return value
: A reference to the newly created window object.

DISCUSSION

Your part calls this method to re-create the windows of an existing document. You should create the platform-specific window as invisible. After calling this method, you should call the `Show` method of the returned window to make it visible.

This method initializes the reference count of the returned window. When you have finished using that window, you should call its `Release` method.

EXCEPTIONS

`kODErrCannotCreateWindow`
: There is not enough memory to create the new window object.

SEE ALSO

The `ODWindow::Show` method (page 809).
The `ODWindowState::RegisterWindow` method (page 835).

SetBaseMenuBar

Document shell

The `SetBaseMenuBar` method installs the specified menu bar object as the base menu for parts to copy.

```
void SetBaseMenuBar (in ODMenuBar theMenuBar);
```

`theMenuBar` A reference to the menu bar object to be installed as the base menu bar.

DISCUSSION

The base menu bar on the Mac OS platform contains the Apple, Document, Edit, Help, and Application menus.

SEE ALSO

The `ODWindowState::CopyBaseMenuBar` method (page 826).

SetDefaultWindowTitles

Document shell

The `SetDefaultWindowTitles` method synchronizes window titles with the filename of the document of the specified draft.

```
void SetDefaultWindowTitles (in ODDraft draft);
```

`draft` A reference to the open draft object.

PART 2

Types and Constants

Contents

This part describes the OpenDoc types and constants. Under each topic, the scalar types, enumerations, and structures are listed and described alphabetically. Descriptions of most scalar types include descriptions of the constants of that type. If a large number of constants are defined for a particular type, however, these constants are described in their own subsection.

The following naming conventions can help you understand the function of the various OpenDoc identifiers.

- Names of OpenDoc data types, including structures and enumerations, begin with the prefix `OD`.
- Names of OpenDoc constants begin with the prefix `kOD`.
- The names of certain constants include a type prefix after the standard `kOD` prefix. For example, constant position codes begin with `kODPos`, where `kOD` is the standard constant prefix and `Pos` indicates that the constant specifies a position code.
- Names of other constants include a type suffix. For example, the constant transform types end with the suffix `Xform`.

General

This section describes the general types and constants used widely in OpenDoc.

Numeric Data

The following types represent numeric data.

`ODFixed` A 32-bit fixed-point value used to represent noninteger numbers in the range [-32768, 32768). The high 16 bits (including a sign bit) represent the integer part, and the low 16 bits represent a fractional part. In effect, the "binary point" is in the middle of the number.

You can convert an integer to `ODFixed` by shifting it left 16 bits. You can round an `ODFixed` value to an integer by adding 0x8000 (0.5) and shifting the result right 16 bits. You can

convert between `ODFixed` and floating-point by multiplying or dividing by 65536.0. You can add and subtract `ODFixed` values as though they were integers; however, you cannot multiply or divide them directly. The OpenDoc ODMath utility library contains functions for working with the `ODFixed` type.

On the Mac OS, this type is identical to the `Fixed` type. The Mac OS has several Toolbox routines, such as `FixMul` and `FixDiv`, for doing arithmetic on `Fixed` values.

`ODFloat` A floating-point value; the size of this type is platform-dependent.

`ODFract` A 32-bit fixed-point value used to represent noninteger numbers in the range [-2, 2). The high 2 bits (including the sign bit) represent the integer part, and the low 30 bits represent a fractional part.

You can convert an integer to `ODFract` by shifting it left 30 bits. You can round an `ODFract` value to an integer by adding 0x20000000 (.5) and shifting the result right 30 bits. You can convert between `ODFract` and floating-point numbers by multiplying or dividing by a scaling factor of 1073741824.0. You can add and subtract `ODFract` values as though they were integers; however, you cannot multiply or divide them directly. The OpenDoc ODMath utility library contains functions for working with the `ODFract` type.

On the Mac OS, this type is identical to the `Fract` type. The Mac OS has several Toolbox routines, such as `FracMul` and `FracDiv`, for doing arithmetic on `Fract` values.

`ODSLong` A signed 32-bit integer value.

`ODSShort` A signed 16-bit integer value.

Characters, Strings, and Tokens

The following types and constants represent characters, text strings, and tokens created from strings. Types and constants marked [M] are specific to the Mac OS platform.

`ODISOStr` A pointer to an ISO string, that is, a string composed of ASCII characters, terminated by a null character (zero byte). Because the first null character terminates the string, null characters may not be embedded.

`ODIText` A platform-specific structure representing a user-visible international text string. The characters in an international text string are represented by 8-bit bytes; thus a total of 256 byte values can be used. In contrast, only the 128 ASCII characters can be used in an ISO string. On the Mac OS, this type is defined as follows:

```
struct ODIText {
   ODITextFormat  format;
   ODByteArray    text;
};
```

Field descriptions

`format` The format of the text. Currently, `kODTraditionalMacText` is the only format supported on the Mac OS platform.

`text` The text string, represented as an `ODByteArray` structure (page 847) in the specified format.

In the `kODTraditionalMacText` format, the buffer of this byte array contains two 16-bit values followed by the raw text. The first 16-bit value is the script code, the second is the language code. The `_length` field of this byte array indicates the entire length of the buffer in 8-bit bytes;

subtract the length of the two codes (4 bytes) to get the number of characters in the text string.

Related constants

`kODISO10646_1993BaseEncoding`
An 18-bit code indicating the document interchange format of text written out by OpenDoc. This fully decomposed format corresponds to ISO standard 10646-1, 1993; it is the only interchange format for text currently supported by OpenDoc.

When international text is stored in an interchange format, the interchange-format code is stored with the encoded text.

`ODITextFormat`
A 32-bit value specifying a text format. The only format currently supported on the Mac OS platform is `kODTraditionalMacText`; more formats may be supported in the future.

Constants of this type

`kODTraditionalMacText` [M]
The traditional Mac OS international text standard of script code/language code/string.

`ODName` A name in international text, of type `ODIText` (page 845).

`ODSByte` A signed 8-bit value, typically used to represent a single character.

`ODType` A string of type `ODISOStr` (page 845) used generically within OpenDoc to represent drag-image types, object types, and value types for storage units, frame presentation types and view types, focus types, and extension types. The constants defined for this data type are described under the sections

related to data transfer, storage units, user interface, and extensions.

`ODTypeToken` A 32-bit value used to represent the tokenized form of an `ODType` value.

Constants of this type

`kODNullTypeToken`	A null type token.
`kODNullFocus`	No focus. (This value is returned by focus iterators.)

You can call the session object's `Tokenize` method (page 598) to convert an `ODType` value into an `ODTypeToken` value.

Time

The following platform-independent type represents time.

`ODTime` A 32-bit value representing a point in time as the number of seconds elapsed since midnight, January 1, 1970.

Arbitrary Data

The following types represent arbitrary data of various sizes.

`ODByteArray` A structure representing foreign data larger than 4 bytes, or variable-length data in general.

```
struct ODByteArray {
   ODULong       _maximum;
   ODULong       _length;
   ODUByte     * _buffer;
};
```

(The SOMobjects™ IDL compiler may generate types equivalent to `ODULong` and `ODUByte` for the fields of this structure.)

Field descriptions

`_maximum`	The size (in bytes) of the memory block containing data. This field indicates the maximum size of data that can be stored there.
`_length`	The length (number of bytes) of the data currently in the memory block.
`_buffer`	A pointer to the memory block containing the data; this memory block is called the byte array's buffer.

In most cases, `ODByteArray` structures are used to pass raw data between OpenDoc and its clients. If a method expects a specific structure in a byte array's buffer, the method description explains what structure the buffer should contain.

`ODUByte` An unsigned 8-bit value.

`ODULong` An unsigned 32-bit value.

`ODUShort` An unsigned 16-bit value.

General Programming Concepts

The following types and constants correspond to frequently used programming concepts.

`ODBoolean` A Boolean value; the size of this type is platform-dependent.

Constants of this type

`kODTrue`	True.
`kODFalse`	False.

`ODError` A 32-bit exception code. The section "Error Codes" on page 904 describes the constants defined for this type.

`ODException` A structure describing an exception.

```
struct ODException {
   ODError        error;
   char           message[256];
};
```

Field descriptions

`error`	The error code identifying the exception.
`message`	A string, used only for debugging purposes, that contains additional information about the exception.

`ODFlags` An unsigned 32-bit value used to represent a collection of flags.

`ODPtr` A general-purpose pointer (that is, a pointer of type `void*`).

`ODSize` An unsigned 32-bit integer value used to specify the size of a data type, a buffer, or a memory block.

The following constant is used throughout OpenDoc.

`kODNULL` A 32-bit value representing null. This constant can be used for a null reference to an OpenDoc object of any class and for null values of most 32-bit data types.

The following constants must be used for null values of the indicated types: `kODNullTypeToken` (page 847) for the `ODTypeToken` type; `kODNullFocus` (page 847) for a null tokenized focus type; `kODNULLKey` (page 873) for the `ODStorageUnitKey` type; `kODNULLID` (page 869) for the types `ODID`, `ODDocumentID`, `ODDraftID`, and `ODStorageUnitID`; and `kODNoEditor` (page 899) for the `ODEditor` type.

Layout

This section describes the types and constants used when parts lay themselves out for display.

Icons

The following type represents a family of icons, including 16 by 16, 32 by 32, and so forth.

`ODIconFamily` An opaque platform-specific type that contains a collection of icons for rendering a part or other content in icon form. On the Mac OS, this type is the same as an `IconSuite` handle.

The following 16-bit constants specify the size of icons, expressed as the number of pixels in either dimension of the icon.

`kODLargeIconSize` The size of a large icon, which is 32 by 32 pixels.

`kODSmallIconSize` The size of a small icon, which is 16 by 16 pixels.

`kODThumbnailSize` The size of a thumbnail icon, which is 64 by 64 pixels.

`kODTinyIconSize` The size of a tiny icon, which is 12 by 12 pixels.

Facets

The following types and constants describe nongeometric information about facets.

`ODHighlight` An enumeration specifying the possible highlight states of a facet.

Constants of this type

`kODDimHighlight`	Highlighted in background style.
`kODFullHighlight`	Highlighted in foreground style.
`kODNoHighlight`	Not highlighted.

`ODSiblingOrder`

An enumeration specifying the order in which siblings are processed when iterating through a facet hierarchy.

Constants of this type

`kODBackToFront`	From back to front.
`kODFrontToBack`	From front to back.

`ODTraversalType`

An enumeration specifying the order of iteration through a facet hierarchy. The root of the hierarchy is the facet whose embedded facets are being traversed. If that facet's sibling order is front to back, sibling facets at the same level in the hierarchy are traversed starting with the frontmost; if the sibling order is back to front, sibling facets are traversed starting with the backmost.

Constants of this type

`kODBottomUp`	Traverse the facet hierarchy bottom up, visiting a parent facet after visiting all its children. If sibling order is front to back, traversal starts with the frontmost facet at the lowest level in the hierarchy; if back to front, at the backmost facet at the lowest level.
`kODChildrenOnly`	Traverse only the children of the root facet.
`kODTopDown`	Traverse the facet hierarchy top down, in depth-first order.

Frames

The following types and constants describe nongeometric information about frames.

`ODFramePosition`

An enumeration specifying the possible positions of a frame relative to a sibling frame.

Constants of this type

`kODFrameBehind`	This frame is behind its sibling.
`kODFrameInFront`	This frame is in front of its sibling.

`ODLinkStatus`

An enumeration specifying the link status of a frame.

Constants of this type

`kODInLinkDestination`	The frame is embedded in the destination of a link; the content of this frame is thus supplied by a link.
`kODInLinkSource`	The frame is embedded in content that is the source of one or more links, but not in content that is the destination of a link.
`kODNotInLink`	The frame is not embedded in any linked content, source or destination.

The following constants of type `ODType` (page 846) specify frame view types and presentations. OpenDoc methods use the tokenized strings for view types and presentations. You can call the session object's `Tokenize` method (page 598) to obtain a token corresponding to one of these constants.

Presentations

`kODPresDefault`	The default presentation for a frame.

View types

`kODViewAsFrame`	Frame.
`kODViewAsLargeIcon`	Large icon (standard icon).
`kODViewAsSmallIcon`	Small icon.
`kODViewAsThumbnail`	Thumbnail icon.

Part Info

The following type represents part-specific data stored with a frame or a facet.

`ODInfoType` An opaque type for the part info data. To use the data, you should cast an `ODInfoType` value to and from a pointer to your part's own representation of the data.

Drawing

This section describes the types and constants used when parts draw themselves.

Basic Imaging

The following types and constants are used frequently in drawing operations. Types marked [M] are specific to the Mac OS platform.

`ODGraphicsSystem`
A 16-bit value specifying a native platform graphics system. Every graphics system supported by OpenDoc should be identified by a unique `ODGraphicsSystem` value. These values are used by transform objects and shape objects to tag graphics-system-specific data.

Constants of this type	
`kODNoGraphicsSystem`	No graphics system.
`kODQuickDraw`	The QuickDraw graphics system.
`kODQuickDrawGX`	The QuickDraw GX graphics system.

`ODPlatformCanvas`
A 32-bit value identifying a platform-specific or graphics-system-specific drawing environment.

`ODPlatformPrintJob`
A 32-bit value identifying a platform-specific or graphics-system-specific print job data structure.

`ODPlatformShape`
A 32-bit value identifying a graphics-system-specific shape. The format of the shape data is unspecified, and it must be tagged with an `ODGraphicsSystem` value (page 853) to identify the graphics system to which it belongs. Given the pair (`ODPlatformShape`, `ODGraphicsSystem`), it is always possible to identify the exact type of the shape data.

`ODgxShape` [M]
A 32-bit value representing a QuickDraw GX shape object; this type is identical to the QuickDraw GX type `gxShape`.

`ODRgnHandle` [M]
A 32-bit value representing a handle to a QuickDraw region; this type is identical to the QuickDraw type `RgnHandle`.

Geometry

The following types and constants represent geometric structures, such as distances, positions, bounding rectangles, and shapes. At runtime, shapes are

usually represented by objects of the `ODShape` class (page 606), but can be accessed and stored in the forms described here.

`ODCoordinate`

A 32-bit value representing a spatial coordinate in a window or document. By default, OpenDoc represents a coordinate with 16 integer bits and 16 fractional bits, which is identical to a value of the `ODFixed` type (page 843). You can partition the bits of an `ODCoordinate` value in any way, provided that you shift the values appropriately when passing information outside OpenDoc. (If you are using a graphics system that handles arbitrary transformations, such as QuickDraw GX, you can do this automatically by assigning a scaling factor to your internal transform.)

`ODPoint` A structure representing a spatial point in a window or document.

```
struct ODPoint {
   ODCoordinate   x;
   ODCoordinate   y;
};
```

Field descriptions

`x`	The x coordinate.
`y`	The y coordinate.

In two-dimensional imaging models (the only imaging models that currently exist for OpenDoc), a point is represented as a pair of `ODCoordinate` values (page 855).

On the Mac OS, this type is identical to the QuickDraw GX `gxPoint` type. Developers who are accustomed to QuickDraw should note that the x coordinate appears first and that the coordinates are fixed-point numbers.

`ODRect` A structure representing a rectangle whose sides are aligned with the axes of the current coordinate system.

```
struct ODRect {
   ODCoordinate   left;
   ODCoordinate   top;
   ODCoordinate   right;
   ODCoordinate   bottom;
};
```

Field descriptions

`left`	The left coordinate of the rectangle.
`top`	The top coordinate of the rectangle.
`right`	The right coordinate of the rectangle.
`bottom`	The bottom coordinate of the rectangle.

A rectangle is represented as four `ODCoordinate` values (page 855). By convention, a rectangle does not include its bottom or right edge; this makes it easier to have adjacent yet nonoverlapping rectangles.

On the Mac OS, this type is identical to the QuickDraw GX `gxRect` type. Developers who are accustomed to QuickDraw should note that the left coordinate comes before the top one, and that the right coordinate comes before the bottom one.

`ODContour` A closed loop of three or more points connected by straight edges. A contour is represented as a 32-bit signed value, indicating the number of points in the contour, followed by the specified number of `ODPoint` structures (page 855) representing the individual points.

The order of points in a contour is significant: for a left-oriented coordinate system like QuickDraw, points arranged in a clockwise order represent a positive area, whereas in the opposite order they represent a negative area or hole (in a right-oriented coordinate system such as OS/2, the reverse is true).

An `ODContour` structure exists only as a component of an `ODPolygon` structure (page 856).

`ODPolygon` A structure of type `ODByteArray` (page 847) representing a two-dimensional shape composed of one or more contours.

Polygons with multiple contours may be composed of disjoint pieces or may have interior holes. The buffer of a polygon byte array contains a 32-bit signed value, indicating the number of contours in the polygon, followed by the specified number of `ODContour` structures (page 856) representing the individual contours.

The `ODPolygon` structure is the platform-independent interchange format for all shapes, including frame shapes.

Shapes and Transforms

The following types and constants relate to shapes and transforms.

`ODGeometryMode`

An enumeration specifying the geometry modes of a shape object, indicating whether the shape is required to maintain its geometric (polygonal) representation. Shapes (such as frame shapes) that are stored persistently in storage unit values must be stored as polygons. A shape loses its polygonal representation if it is combined with (that is, unioned with, subtracted from, or intersected with) a shape that does not have polygonal representation.

Constants of this type

`kODLoseGeometry`	The shape does not need to use a polygon to describe its geometric representation. The polygonal representation can be discarded in order to optimize speed, at the expense of accuracy and persistent storage capability. A facet's clip shape will generally have this mode.
`kODNeedsGeometry`	The shape must maintain its polygonal representation. A facet's frame shape and used shape will have this mode because they are stored persistently in polygonal form.

`kODPreserveGeometry`
: The shape must maintain its polygonal representation for as long as possible. This is the default value.

`ODMatrix`
: A structure representing a platform-independent coordinate transformation.

```
struct ODMatrix {    // Note: Rightmost column
   ODFixed m[3][3];  //    of matrix m contains
};                   //    ODFract elements.
```

Field descriptions

`m`
: A 3-by-3 transform matrix of fixed-point numbers.

Although the transform matrix in the `m` field is declared to be of the `ODFixed` type (page 843), the rightmost column of the matrix (elements `m[0][2]`, `m[1][2]`, and `m[2][2]`) are actually of the `ODFract` type (page 844).

A transform matrix can be used for translation, scaling, skewing, rotation, or any combination. See the `ODTransform` class description (page 736) for more details.

The `ODMatrix` structure is equivalent to the QuickDraw GX `gxMapping` structure.

`ODTransformType`
: A 16-bit value used to specify the type of transformation specified by a transform object.

Constants of this type

`kODIdentityXform`
: Identity (no-op) transform.

`kODLinearTranslateXform`
: Scale, rotate, skew, and translation.

`kODLinearXform`
: Scale, rotate, and skew.

`kODPerspectiveXform`
: Perspective transformation, which applies a 3D or distortion effect.

`kODScaleTranslateXform`	Scale and translation.
`kODScaleXform`	Pure scale.
`kODTranslateXform`	Pure translation (offset).

User Events

This section describes the types and constants used in handling user events.

Focus Types

The following type and constants represent focus types that frames may own.

`ODFocusType` A string of type `ODType` (page 846) used to identify a focus type.

Constants of this type

`kODClipboardFocus`	The frame with the clipboard focus has access to the clipboard.
`kODKeyFocus`	The frame with the keyboard focus receives keyboard events (excluding page-movement key events).
`kODMenuFocus`	The frame with the menu focus receives menu events.
`kODModalFocus`	The frame with the modal focus is notifying other frames that it is the only currently modal frame.
`kODMouseFocus`	The frame with the mouse focus receives all mouse-up and mouse-down events; it also receives mouse-within events independent of the facet in which the cursor is located.

`kODScrollingFocus`	The frame with the scrolling focus receives page-movement key events (such as Page Up). This frame need not own keyboard focus.
`kODSelectionFocus`	The frame with the selection focus receives modified mouse-click events (Shift-click and Command-click). OpenDoc draws the active frame border around all facets of this frame.

OpenDoc methods use the tokenized form of the focus type constants. You can call the session object's `Tokenize` method (page 598) to obtain a token corresponding to one of these constants.

Events

The following types represent information about user events.

`ODEventData` A platform-specific structure representing an event. On the Mac OS, it is defined as a Mac OS event record.

```
struct EventRecord {
   short          what;
   long           message;
   long           when;
   Point          where;
   short          modifiers;
};
```

Field descriptions

`what`	The type of event received.
`message`	Additional information associated with the event.
`when`	The time when the event was posted.

`where`	For mouse events, the location of the cursor (in global coordinates) at the time the event was posted. This field is a QuickDraw `Point` structure, not an `ODPoint` structure.
`modifiers`	Information about the state of the modifier keys and the mouse button at the time the event was posted. For activate events, this field also indicates whether the window should be activated or deactivated.

`ODEventInfo` A structure containing context-specific event information.

```
struct ODEventInfo {
   ODFrame        embeddedFrame;
   ODFacet        embeddedFacet;
   ODPoint        where;
   ODBoolean      propagated;
};
```

Field descriptions

`embeddedFrame`	The frame within which the event occurred. Only relevant for events that are delivered to the containing part.
`embeddedFacet`	The facet within which the event occurred. Only relevant for events that are delivered to the containing part.
`where`	The position in local (frame) coordinates where the event occurred. Only relevant for mouse-related events.
`propagated`	`kODTrue` if the event occurred in an embedded frame that propagates events and if that frame's part did not handle the event.

`ODEventType` A 16-bit platform-specific code specifying an event type. The section "Event Types" on page 862 describes event-type constants defined for the Mac OS platform.

`ODIdleFrequency`
A 32-bit unsigned number representing the frequency, in ticks or 60ths of a second, at which null events are sent during idle times.

Event Types

This section describes the constants of type `ODEventType` (page 862) used on the Mac OS platform to specify the event types; some of these constants may be used on other platforms as well. The constants that correspond to the standard Mac OS event types are equivalent to the standard Mac OS event codes.

Standard Mac OS event types

`kODEvtActivate`	An activate event, which indicates that a window was activated or deactivated.
`kODEvtAutoKey`	An autokey event, which indicates that the user has held down a key for a certain amount of time.
`kODEvtDisk`	A disk event.
`kODEvtKeyDown`	A key-down event.
`kODEvtKeyUp`	A key-up event.
`kODEvtMouseDown`	A mouse-down event.
`kODEvtMouseUp`	A mouse-up event.
`kODEvtNull`	A null event, which indicates idle time.
`kODEvtOS`	An OS event (a suspend/resume event or a mouse-moved event).
`kODEvtUpdate`	An update event.

OpenDoc-specific event types

`kODEvtBGMouseDown` A mouse-down event while the process is inactive.

`kODEvtBGMouseDownEmbedded`
A mouse-down event in an embedded frame while the process is inactive. This event is sent to the containing part.

`kODEvtMenu` An adaptation of a mouse-down event in the menu bar or the equivalent Command-key combination. On the Mac OS, the `message` field of the event record contains the menu in the high word and the item in the low word. Other fields of the event record are the same as for a mouse-down or key-down event.

`kODEvtMouseDownBorder`
A mouse-down event in the active border (around facets of the frame with the selection focus). On the Mac OS, the `message` field of the event record contains the embedded facet. Other fields of the event record are the same as for a mouse-down event.

`kODEvtMouseDownEmbedded`
A mouse-down event in an embedded facet. This event is sent to the containing facet. On the Mac OS, the `message` field of the event record contains the embedded facet. Other fields of the event record are the same as for a mouse-down event.

`kODEvtMouseEnter` Sent when the mouse first enters a frame (with the mouse button up). The `where` field of the `ODEventInfo` structure specifies the mouse location in local (frame) coordinates.

`kODEvtMouseLeave` Sent when the mouse moves out of a frame (with the mouse button up). The `where` field of the `ODEventInfo` structure specifies the mouse location in local (frame) coordinates.

`kODEvtMouseUpEmbedded`
: A mouse-up event in an embedded facet. This event is sent to the containing facet. On the Mac OS, the `message` field of the event record contains the embedded facet. Other fields of the event record are the same as for a mouse-up event.

`kODEvtMouseWithin`
: Sent when the mouse moves within a frame (with the mouse button up). An `ODEventInfo` structure specifies the mouse location in local (frame) coordinates.

`kODEvtWindow`
: An adaptation of a mouse event in the title bar of a window. On the Mac OS, the `message` field of the event record contains the part code returned by the `FindWindow` routine. Other fields of the event record are the same as for a mouse-down event.

Mouse Location

The following constants of type `ODSShort` (page 844) classify the location where a mouse-up or mouse-down event occurred; they are equivalent to the codes returned by the Mac OS routine `FindWindow`. OpenDoc uses these codes internally to decide how to dispatch mouse events. Constants marked [M] are specific to the Mac OS platform.

`kODMDInContent` [M]
: In the content region of an application window.

`kODMDInDesk` [M]
: In the desktop.

`kODMDInDrag` [M]
: In the drag region.

`kODMDInGoAway` [M]
: In the close box of a window.

`kODMDInGrow` [M]
: In the size box of a window.

`kODMDInMenuBar` [M]
: In the menu bar.

`kODMDInSysWindow` [M]
: In a system window.

`kODMDInZoomIn` [M]
: In the zoom box of a zoomed-out window.

`kODMDInZoomOut` [M] In the zoom box of a zoomed-in window.

Windows and Menus

This section describes the types and constants used in handling windows and menus.

Windows

The following type represents a window structure.

`ODPlatformWindow` A 32-bit value identifying a platform-specific window. On the Mac OS, this type is identical to the `WindowPtr` type.

The following constant identifies a window resource; it is specific to the Mac OS platform.

`kODPaletteWDEFID` [M] The resource ID for the `'WDEF'` resource for a floating window.

Menus

The following types are used for handling menus. Types marked [M] are specific to the Mac OS platform.

`ODCommandID` [M]
A 32-bit value representing the command ID associated with a particular menu/item combination. The section "Menu Command IDs" on page 866 describes the constants defined for this type.

`ODMenuID` A platform-specific identifier for a menu; on the Mac OS, an identifier of type `ODSShort` (page 844).

`ODMenuItemID`
: A platform-specific identifier for a menu item; on the Mac OS, an identifier of type `ODSShort` (page 844).

`ODPlatformMenu`
: A 32-bit value identifying a platform-specific menu; on the Mac OS, this type is identical to the `MenuHandle` type.

`ODPlatformMenuBar`
: A 32-bit value identifying a platform-specific menu bar; on the Mac OS, this type is identical to the `Handle` type.

Menu Command IDs

This section describes constants of type `ODCommandID` (page 865). Most of these constants define the position-independent command IDs for the standard menus and menu items in the base menu bar, as provided by the OpenDoc shell. The remaining constants are used to specify legal ranges for command IDs. Constants marked [M] are specific to the Mac OS platform.

Menus

`kODCommandAppleMenu` [M] The Apple menu.

`kODCommandDocumentMenu` [M] The Document menu.

`kODCommandEditMenu` [M] The Edit menu.

Apple menu items

`kODCommandAbout` [M] About *Editor*, where *Editor* is the name of the active part editor.

Document menu Items

`kODCommandClose` [M] Close.

`kODCommandDeleteDocument` [M] Delete Document.

`kODCommandDocumentInfo` [M] Document Info.

`kODCommandDraft` [M]	Drafts.
`kODCommandInsert` [M]	Insert.
`kODCommandNew` [M]	New.
`kODCommandOpen` [M]	Open Selection; applies to a selected frame.
`kODCommandOpenDocument` [M]	Open Document.
`kODCommandPageSetup` [M]	Page Setup.
`kODCommandPrint` [M]	Print.
`kODCommandRevert` [M]	Revert to Saved.
`kODCommandSave` [M]	Save.
`kODCommandSaveACopy` [M]	Save a Copy.

Edit menu items

`kODCommandClear` [M]	Clear.
`kODCommandCopy` [M]	Copy.
`kODCommandCut` [M]	Cut.
`kODCommandGetPartInfo` [M]	*Selection* Info, where *Selection* is "Part" if an embedded frame is selected, "Link" if a link border is selected, or a brief part-specific description if the part's intrinsic content is selected.
`kODCommandPaste` [M]	Paste.
`kODCommandPasteAs` [M]	Paste As.
`kODCommandPreferences` [M]	*Editor* Preferences, where *Editor* is the name of the active part editor.
`kODCommandRedo` [M]	Redo.

`kODCommandSelectAll` [M]	Select All.
`kODCommandUndo` [M]	Undo.
`kODCommandViewAsWin` [M]	View in Window.

Menu ID ranges

`kODCommandShellFirst` [M]	Lower limit for command IDs defined by the document shell or a container application.
`kODCommandShellLast` [M]	Upper limit for command IDs defined by the document shell or a container application; lower limit for command IDs defined by part editors.

Undo/Redo Actions

The following types and constants represent information about undo/redo actions.

`ODActionData`

A structure of type `ODByteArray` (page 847) whose buffer contains the action data for Undo/Redo commands.

`ODActionType`

An enumeration specifying the possible values for an undo action.

Constants of this type

`kODBeginAction`	The action was the first of a multistep action (such as the drag part of a drag-and-drop action).
`kODEndAction`	The action was the last of a multistep action (such as the drop part of a drag-and-drop action).
`kODSingleAction`	The action was a single action.

`ODDoneState` An enumeration specifying the state of an undo action.

Constants of this type

`kODDone`	The action was done and is now on the undo stack.
`kODRedone`	The action was done, undone, and redone and is now back on the undo stack.
`kODUndone`	The action was done and undone and is now on the redo stack.

`ODRespectMarksChoices`
An enumeration specifying the possible values for clearing an action history.

Constants of this type

`kODDontRespectMarks`	Clear the whole action history, ignoring any marks that indicate the beginning of an action subhistory.
`kODRespectMarks`	Clear only the actions within the current subhistory.

Storage

This section describes the types used by the OpenDoc storage system.

Object IDs

The following types and constants are used in many storage operations.

`ODID` A 32-bit value used as an identifier for a particular type of object (for example, a document or a storage unit).

Constants of this type

`kODNULLID`	A null ID or an invalid ID.

`kODIDAll`	Any ID. This value is used when specifying which storage units (or other objects) are of interest to a particular operation.
`kODIDWild`	Any ID. This value is used when specifying which storage units (or other objects) are of interest to a particular operation.

Note that these three constants have the same value; their different names provide clarity to the code that uses them. Typically `kODNULLID` is used to mean an absent or invalid ID. `kODIDAll` or `kODIDWild` are used to specify no restrictions on the scope of a cloning operation.

`ODPersistentObjectID`
A 32-bit identifier, used for scripting, of a part or frame.

Container Suites and Storage Containers

The following types and constants relate to container suites. Types and constants marked [M] are specific to the Mac OS platform.

`ODContainerID`
A structure of type `ODByteArray` (page 847) identifying a container. The buffer of this byte array holds a container-suite-specific identifier for a container. The structure of the buffer depends on the type of container. For example, the identifier for a file container specifies a file-system file; the identifier for a memory container is a handle for a relocatable memory block.

`ODContainerName`
A user-readable name of type `ODName` (page 846) for a container object.

`ODContainerSuite`
An opaque type representing a specific container suite.

`ODContainerType`

A string of type `ODISOStr` (page 845) used to specify a type of storage container.

Constants of this type

`kODBentoFileContainer`
The Bento container class for documents.

`kODBentoMemoryContainer`
The Bento container class for drag and drop and the clipboard.

`kODDefaultFileContainer`
The default container type for documents on this platform.

`kODDefaultMemoryContainer`
The default container type for drag and drop and the clipboard on this platform.

On each platform, OpenDoc has a default container type that it uses for documents and a default container type that it uses for drag and drop and the clipboard. On the Mac OS, these defaults are the Bento file container and the Bento memory container, respectively.

`ODOSType` [M] A 32-bit wrapper for a Mac OS `OSType` structure, which specifies a Mac OS file type.

Constants of this type

`kODShellSignature` [M]
The creator type for the OpenDoc document shell (value `'odtm'`).

`kODNameMappings` [M]
The resource type designation for the name-mapping resource (value `'nmap'`).

Documents

The following types and constants relate to OpenDoc documents.

`ODDocumentID`
: An identifier of type `ODID` (page 869) for a document. Use the `ODID` constant `kODNULLID` to represent a null document ID.

Constants of this type

`kODDefaultDocument`
: The default document ID.

`ODDocumentName`
: A user-readable name of type `ODName` (page 846) for a document.

Drafts

The following types and constants relate to drafts.

`ODDraftID` An identifier of type `ODID` (page 869) for a draft. Use the `ODID` constant `kODNULLID` to represent a null draft ID.

`ODDraftKey` A 32-bit value used as an identifier for a cloning transaction.

`ODDraftName` A name of type `ODISOStr` (page 845) for a draft.

`ODDraftPermissions`
: An enumeration specifying the possible values for draft permissions.

Constants of this type

`kODDPExclusiveWrite`
: Exclusive read/write access.

`kODDPNone` No access.

`kODDPReadOnly` Read-only access.

`kODDPSharedWrite` Shared read/write access.

`kODDPTransient` Navigation-only access.

The Bento container suite supports only the `kODDPReadOnly` and `kODDPExclusiveWrite` draft permissions.

Storage Units

The following types and constants relate to storage units, which are used for storage and data transfer.

`ODObjectType`

A string of type `ODType` (page 846) specifying the type of persistent object represented by a particular storage unit.

Constants of this type

`kODFrameObject`	A frame.
`kODNonPersistentFrameObject`	A nonpersistent frame.
`kODPartObject`	A part.

`ODStorageUnitID`

An identifier of type `ODID` (page 869) for a storage unit. Use the `ODID` constant `kODNULLID` to represent a null storage-unit ID.

`ODStorageUnitKey`

A 32-bit value used as an access key for locking and unlocking a storage unit.

Constants of this type

`kODNULLKey`	A storage unit key has not yet been granted or could not be granted.

`ODStorageUnitName`

A name of type `ODISOStr` (page 845) for a storage unit.

`ODStorageUnitRef`

An opaque type representing a persistent reference for a storage unit. Whereas a value of the `ODStorageUnitID` type identifies a storage unit within the current session, a value of the `ODStorageUnitRef` type identifies it persistently across sessions.

Related constants

`kODStorageUnitRefSize`
: The size (number of bytes) of an `ODStorageUnitRef`.

A value of the `ODStorageUnitRef` type is created by a storage unit, which must be focused on the value where the persistent reference will be stored. The scope of the persistent reference is limited to the value for which it was created; if you store it in a different value, it will almost certainly not refer to the correct storage unit. For more information on persistent references to storage units, see the chapter on storage in the *OpenDoc Programmer's Guide for the Mac OS*.

Properties and Values

The following types and constants relate to the properties and values in a storage unit.

`ODPropertyName`
: A name of type `ODISOStr` (page 845) for a property within a storage unit. The section "Property Names" on page 875 describes the constants defined for this type.

`ODValueIndex`
: A 32-bit integer used as an index for a value within a property of a storage unit. The first value created for a property has index 1; the second, 2; and so on.

Constants of this type

`kODIndexAll`
: Any value index. This constant is used when focusing a storage unit on a property and when checking for the existence of a property in a storage unit.

`ODValueType`
: A string of type `ODType` (page 846) used to identify the type of data in the value of a storage unit. The section "Value Types" on page 882 describes the constants defined for this type.

Property Names

This section describes the constants defined for the `ODPropertyName` type (page 874). These constants are names of the standard OpenDoc properties used in storage units, or prefixes used to construct property names.

Prefixes

The following constants are prefixes in property names; they specify the type of information stored in the property.

`kODPropPreAnnotation` The prefix in the names of all annotation properties. An annotation is a property that should be copied automatically whenever the storage unit is cloned. Part-editor developers can use this string as a prefix in the names of annotation properties they define.

`kODPropPreODMetaData` The prefix in the names of all OpenDoc metadata properties. Metadata is information about the data itself, such as the time it was last modified. Part-editor developers should not use this prefix.

Draft Properties

The following properties save the specified information about the draft. Properties marked [M] are specific to the Mac OS platform.

`kODPropDraftComment` Any user-created comments on the draft; value type `kODMacIText`.

`kODPropDraftNumber` The number of this draft; value type `kODSLong`.

`kODPropDraftSavedDate`
The draft-saved date for drafts that were saved via the Draft History dialog box; value type `kODTime_T`.

`kODPropEditionID` [M] The last edition file ID used by the current draft of the document; value type `kODULong`.

`kODPropSectionID` [M] The last section ID used by the current draft of the document; value type `kODULong`.

`kODPropRootFrameList` A list of strong references to the root frames of saved windows; value type `kODStrongStorageUnitRefs`.

`kODPropRootPartSU` A strong reference to the root part of this draft; value type `kODStrongStorageUnitRef`.

Persistent-Object Properties

The following properties are used by persistent objects of all kinds.

`kODPropObjectType` The type of persistent object (for example, part or frame); value type `kODISOStr`.

`kODPropStorageUnitType`
The type of storage unit (for example, the storage unit for a frame object, for link content, and so forth); value type `kODISOStr`.

Part Properties

The following properties save the specified information about parts.

`kODPropComments` Any user-created comments about the part; value type `kODMacIText`.

`kODPropContents` The stored intrinsic data for the part; the value types correspond to part kinds supported by the part.

`kODPropCreateDate` The date and time when the part was created; value type `kODTime_T`.

`kODPropCustomIcon` The custom icon for a part; value type `kODIconFamily`. If the part has a custom icon, the custom icon is stored in this property.

`kODPropDisplayFrames` A list of weak references to the display frames of this part; value type `kODWeakStorageUnitRefs`.

`kODPropIsStationery` Specifies whether this part is a stationery part; value type `kODBoolean`.

`kODPropModDate` The date and time when the part was last modified; value type `kODTime_T`.

`kODPropModUser` The user name when the part was last modified; value type `kODMacIText`.

`kODPropName` The name of the part; value type `kODMacIText`.

`kODPropPageSetup` Page setup information to be used by a root part running a print job. The value type is platform-specific; on the Mac OS, either `kODTypeQuickDrawPageSetup` or `kODTypeGXPageSetup`.

`kODPropPreferredEditor` The editor ID of the preferred editor (the editor that last wrote this part to persistent storage); value type `kODEditor`.

`kODPropPreferredKind` The value in the `kODPropContents` property that should be read by an editor bound to this part; value type `kODISOStr`.

Frame Properties

The following properties save the specified information about frames.

`kODPropBiasTransform` The bias transform to be attached to the canvas on which this frame is drawn; value type `kODTransform`.

`kODPropContainingFrame` A weak reference to the containing frame of this frame; value type `kODWeakStorageUnitRef`.

`kODPropDoesPropagateEvents`
: Specifies whether this frame's part propagates events; value type `kODBoolean`.

`kODPropFrameGroup`
: The group ID of the frame group to which this frame belongs; value type `kODULong`.

`kODPropFrameShape`
: The frame shape for this frame; value type `kODPolygon`.

`kODPropGraphicsSystem`
: The graphics system used to draw in this frame's part; value type `kODSShort`. The stored data is interpreted as type `ODGraphicsSystem` (page 853).

`kODPropInternalTransform`
: The internal transform for this frame; value type `kODTransform`.

`kODPropIsFrozen`
: Specifies whether this frame is bundled; value type `kODBoolean`.

`kODPropIsOverlaid`
: Specifies whether this frame is overlaid; value type `kODBoolean`.

`kODPropIsRoot`
: Specifies whether this frame is the root frame of a window; value type `kODBoolean`.

`kODPropIsSubframe`
: Specifies whether this frame is a subframe of another frame; value type `kODBoolean`.

`kODPropLinkStatus`
: The link status of this frame; value type `kODULong`, interpreted as type `ODLinkStatus` (page 852).

`kODPropPart`
: A strong reference to the part displayed in this frame; value type `kODStrongStorageUnitRef`.

`kODPropPartInfo`
: The part info (part-specific data) stored with this frame. The value type is determined by the part editor.

`kODPropPresentation` The presentation of the part displayed in this frame; value type `kODISOStr`.

`kODPropSequenceNumber`
The sequence number of this frame within its frame group; value type `kODULong`.

`kODPropViewType` The view type of the part displayed in this frame; value type `kODISOStr`.

`kODPropWindowProperties`
A strong reference to a storage unit containing window size and so forth; value type `kODStrongStorageUnitRef`. This property is added to the root frame of a saved window.

Window Properties

The following properties save the specified information about windows. Properties marked [M] are specific to the Mac OS platform

`kODPropRootFrame` A strong reference to the root frame; value type `kODStrongStorageUnitRef`.

`kODPropShouldShowLinks`
Specifies whether parts in this window should display link borders; value type `kODBoolean`.

`kODPropSourceFrame` A strong reference to the source frame; value type `kODStrongStorageUnitRef`.

`kODPropWindowHasCloseBox`
Specifies whether this window has a close box; value type `kODBoolean`.

`kODPropWindowHasZoomBox`
Specifies whether this window has a zoom box; value type `kODBoolean`.

`kODPropWindowIsFloating`
: Specifies whether this window can float; value type `kODBoolean`.

`kODPropWindowIsResizable`
: Specifies whether this window has a resize box; value type `kODBoolean`.

`kODPropWindowIsRootWindow`
: Specifies whether this window is a root window; value type `kODBoolean`.

`kODPropWindowIsVisible`
: Specifies whether this window is visible; value type `kODBoolean`.

`kODPropWindowProcID` [M]
: The Mac OS definition ID of the window; value type `kODSShort`.

`kODPropWindowRect`
: The bounding rectangle of a window; value type `kODRect`.

`kODPropWindowRefCon`
: The window's reference constant, which is set by your part; value type `kODSLong`.

`kODPropWindowTitle`
: The title of a window; on the Mac OS, the value type is `kODMacIText`.

Data-Transfer Properties

The following properties save the specified information in the content storage units of data-transfer objects (the clipboard, the drag-and-drop object, link-source objects, and link objects).

General

`kODPropContents`
: The data being transferred in the same format as in the storage unit of the source part; the value types correspond to the types (part kinds) of the data being transferred.

`kODPropContentFrame` A weak reference to the embedded frame being transferred; value type `kODWeakStorageUnitRef`. This property exists only if the data being transferred consists of a single embedded frame.

`kODPropName` If the transferred data is embedded at the destination, the name to be used for the resulting embedded part; value type `kODMacIText`.

`kODPropProxyContents` Suggested adornments to apply to the embedded frame being transferred; value type depends on type of adornment being transferred. This property exists only if the data being transferred consists of a single embedded frame.

`kODPropSuggestedFrameShape` If the transferred data is embedded at the destination, the suggested shape for the resulting embedded frame; the value type is either `kODPolygon` or a platform-specific value type.

`kODPropCloneKindUsed` The kind of cloning operation used to clone objects into this data-transfer object; value type `kODCloneKind`.

Clipboard or drag and drop

`kODPropLinkSpec` A link specification; value type `kODLinkSpec`. This property indicates that the destination part may paste a link to the original content being transferred.

Drag and drop

`kODPropMouseDownOffset` The offset of the location at which a mouse-down event occurred from the top left corner of the selection; value type `kODPoint`.

Link object

`kODPropLinkSource`	A weak reference to the link-source object associated with this link object; value type `kODWeakStorageUnitRef`.

Link-source object

`kODPropAutoUpdate`	Specifies whether this link is to be updated automatically; value type `kODBoolean`.
`kODPropChangeTime`	The date and time when this link was last updated; value type `kODTime_T`.
`kODPropLink`	A weak reference to the link object associated with this link-source object; value type `kODWeakStorageUnitRef`.
`kODPropLinkContentSU`	A strong reference to the content storage unit for the linked data; value type `kODStrongStorageUnitRef`.
`kODPropSourcePart`	A weak reference to the part that contains (or last contained) the source data for this link; value type `kODWeakStorageUnitRef`.

Value Types

This section describes the constants defined for the `ODValueType` type (page 874).

The following value-type constants specify the standard types in which data can be stored in values of storage units. Constants marked [M] are specific to the Mac OS platform.

`kODBoolean`	Type `ODBoolean` (page 848).
`kODCloneKind`	Type `ODCloneKind` (page 887), expressed as a 32-bit value.
`kODEditor`	Type `ODEditor` (page 899).

`kODIconFamily`	Type `ODIconFamily` (page 850).
`kODIconFamilyMac` [M]	Type `ODIconFamily` (page 850) representing a Mac OS icon family.
`kODIconFamilyWin`	Type `ODIconFamily` (page 850) representing a Windows icon family.
`kODIconFamilyOS2`	Type `ODIconFamily` (page 850) representing an OS/2 icon family.
`kODIconFamilyAIX`	Type `ODIconFamily` (page 850) representing an AIX icon family.
`kODIntlText`	Type `ODIText` (page 845) converted to a fully decomposed document-interchange format that corresponds to ISO standard 10646-1, 1993.
`kODISOStr`	Type `ODISOStr` (page 845).
`kODISOStrList`	A list of ISO strings, each of type `ODISOStr` (page 845).
`kODLinkSpec`	Link-specification data for an `ODLinkSpec` object (page 379).
`kODMacIText` [M]	Type `ODIText` (page 845) in traditional Mac OS international text format.
`kODObjectType`	Type `ODObjectType` (page 873).
`kODPoint`	Type `ODPoint` (page 855).
`kODPolygon`	Type `ODPolygon` (page 856).
`kODRect`	Type `ODRect` (page 855).
`kODSLong`	Type `ODSLong` (page 844).
`kODSShort`	Type `ODSShort` (page 844).

`kODStrongStorageUnitRef`
: Type `ODStorageUnitRef` (page 873) representing a strong storage-unit reference.

`kODStrongStorageUnitRefs`
: A list of strong storage-unit references, each of type `ODStorageUnitRef` (page 873).

`kODTime_T`
: Type `ODTime` (page 847).

`kODTransform`
: Type `ODMatrix` (page 858).

`kODTypeGXPageSetup` [M]
: The QuickDraw GX type `gxJob`.

`kODTypeQuickDrawPageSetup` [M]
: The QuickDraw type `THPrint`.

`kODULong`
: Type `ODULong` (page 848).

`kODUShort`
: Type `ODUShort` (page 848).

`kODWeakStorageUnitRef`
: Type `ODStorageUnitRef` (page 873) representing a weak storage unit reference.

`kODWeakStorageUnitRefs`
: A list of weak storage unit references, each of type `ODStorageUnitRef` (page 873).

The following value-type constant is used when focusing a storage unit on a property and when checking for the existence of a property in a storage unit.

`kODTypeAll`
: All value types. This constant is equivalent to `kODNULL`. Their different names provide clarity to the code that uses them. Typically `kODNULL` is used to mean an absent or invalid value type; `kODTypeAll` is used to specify all values of a given property.

Position Codes

Position codes represent either the position of drafts within a document or the position of properties and values within a storage unit.

`ODPositionCode`	A 32-bit value used to specify the position of a draft within a document, or the position of a property or value that defined the focus context for a storage unit.

Draft Position Codes

The following constants of the `ODPositionCode` type represent positions of one draft in a document relative to another draft of the same document. The drafts of a document can be thought of as a stack with the most recent on top; a given draft is said to be above an earlier draft and below a more recent draft.

`kODPosFirstAbove`	The draft above (immediately more recent than) the specified draft.
`kODPosFirstBelow`	The draft below (immediately previous to) the specified draft.
`kODPosLastAbove`	The draft above (immediately more recent than) the specified draft.
`kODPosLastBelow`	The draft below (immediately previous to) the specified draft.
`kODPosSame`	The same draft as the specified draft.
`kODPosTop`	The top (most recent) draft in the document.

Storage-Unit Position Codes

The following constants of the `ODPositionCode` type represent positions of properties and values within a storage unit; they are used to define or to change the focus context of a storage unit.

`kODPosAll`	All properties or all values.

`kODPosFirstSib`	The first property of the storage unit or the first value of the specified property.
`kODPosLastSib`	The last property of the storage unit or the last value of the specified property.
`kODPosMWrap`	Wraps iteration of properties or values.
`kODPosNextSib`	The next property of the storage unit or the next value of the specified property, relative to the current focus context.
`kODPosPrevSib`	The previous property of the storage unit or the previous value of the specified property, relative to the current focus context.
`kODPosSame`	The same property or value context as in the current focus context.
`kODPosUndefined`	Undefined property or value context; typically used when a property name specifies the property and when a value type or value index specifies the value.

Unused Position Codes

The following constants of type `ODPositionCode` are reserved for future use.

`kODPosReserved11`	Reserved for future use.
`kODPosReserved12`	Reserved for future use.
`kODPosReserved13`	Reserved for future use.
`kODPosReserved14`	Reserved for future use.
`kODPosReserved15`	Reserved for future use.

Data Transfer

This section describes the types and constants used during data-transfer operations.

General

The following types and constants are used frequently in data-transfer operations.

`ODUpdateID` A 32-bit value used as an update identifier for clipboard content or linked content. Two `ODUpdateID` values associated with different versions of the same content may be tested for equality, but any other use of these values is meaningless.

Constants of this type

`kODUnknownUpdate`	A value guaranteed to be different from any actual update ID. This constant can be used by parts when the update ID associated with shared content is unknown.

`ODCloneKind` An enumeration specifying the possible semantic values of a clone operation.

Constants of this type

`kODCloneCopy`	Copy to the clipboard object or the drag-and-drop object.
`kODCloneCut`	Cut to the clipboard object or the drag-and-drop object.
`kODCloneDropCopy`	Copy at the destination of a drop.
`kODCloneDropMove`	Move at the destination of a drop.
`kODCloneFromLink`	Clone from a link.
`kODClonePaste`	Paste from the clipboard object.
`kODCloneToLink`	Clone to a link source.

`ODPasteAsMergeSetting`

A 32-bit value used to specify which At the Destination radio button (Merge with Contents or Embed As) is initially selected in the Paste As dialog box and whether the other button is available.

Constants of this type

`kODPasteAsEmbed`	Embed As is initially selected; Merge with Contents is available.
`kODPasteAsEmbedOnly`	Embed As is selected; Merge with Contents is disabled.
`kODPasteAsMerge`	Merge with Contents is initially selected; Embed As is available.
`kODPasteAsMergeOnly`	Merge with Contents is selected; Embed As is disabled.

`ODPasteAsResult`

A structure representing the user's selections in the Paste As dialog box.

```
struct ODPasteAsResult {
   ODBoolean      pasteLinkSetting;
   ODBoolean      autoUpdateSetting;
   ODBoolean      mergeSetting;
   ODTypeToken    selectedView;
   ODType         selectedKind;
   ODType         translateKind;
   ODEditor       editor;
};
```

Field descriptions

`pasteLinkSetting`	`kODTrue` if a link was chosen, otherwise `kODFalse`.
`autoUpdateSetting`	`kODTrue` if automatic updating was chosen, otherwise `kODFalse`. Relevant only if `pasteLinkSetting` is `kODTrue`.

`mergeSetting`	`kODTrue` if incorporation was chosen; `kODFalse` if embedding was chosen.
`selectedView`	The view type chosen. This field contains the tokenized form of a view-type constant (page 853). You can call the session object's `GetType` method (page 587) to convert the token into the corresponding view type.
`selectedKind`	The part kind chosen for merging.
`translateKind`	If data translation was chosen, this field indicates the available type that should be translated to `selectedKind`. If an available kind was chosen, this field is `kODNULL`.
`editor`	The preferred editor to bind to the part after embedding; `kODNoEditor` if no specific editor was chosen. Relevant only if `mergeSetting` is `kODFalse`.

`ODPlatformType`
A 32-bit wrapper for the platform-specific type used to identify data formats for data interchange. On the Mac OS, this type is identical to the `ScrapType` or `OSType` types (a four-character code).

The following constant is used for clipboard and linking operations.

`kODNoWait` An `ODULong` (page 848) constant specifying not to wait when acquiring a busy lock on the clipboard or on a link.

Translation

The following types and constants are used during translation of part data from one part kind to another.

`ODPlatformTypeSpace`
A 32-bit value used to specify the type of a platform-specific structure identifying a type space (data or file).

Constants of this type

`kODPlatformDataType`
The native operating system scrap type.

`kODPlatformFileType`
The native operating system file type.

`ODTranslateResult`
An enumeration specifying the possible results of a translation.

Constants of this type

`kODCannotTranslate`
Translation is not allowed with the given types.

`kODCanTranslate` Translation is allowed with the given types.

Drag and Drop

This section describes the types and constants used for drag-and-drop operations. Types and constants marked [M] are specific to the Mac OS platform.

`ODPlatformDragReference` [M]
A 32-bit value identifying the current drag operation.

The following constant of type `ODType` (page 846) can be passed as a parameter to the `StartDrag` method (page 192) of the drag-and-drop object to

specify the type of image OpenDoc should display to the user as dragging feedback.

`kODDragImageRegionHandle` [M]
: The drag image is a handle to a drag region, as required by the Mac OS Drag Manager.

Drag Attributes

The following constants of type `ODULong` (page 848) are used to represent bit flags that can be set in the drag attributes of a particular drag-and-drop operation. You can inspect the drag attributes by calling the `GetDragAttributes` method (page 187) of the drag-and-drop object; test the returned value for the presence of a particular flag using the bitwise AND operator (for example, the & operator in C++). Some flags are relevant to a part tracking a drag event that has entered one of its facets; others are relevant to the part that is the destination of a drop.

Drag-tracking flags

`kODDragIsInSourceFrame`
: The user has not left the source frame.

`kODDragIsInSourcePart`
: The user has not left the source part.

Drop flags

`kODDropIsInSourceFrame`
: The drop is occurring in the source frame of the drag.

`kODDropIsInSourcePart`
: The drop is occurring in the same part as the source frame of the drag.

`kODDropIsMove` The drag-and-drop operation is a move.

`kODDropIsCopy` The drag-and-drop operation is a copy.

`kODDropIsPasteAs`	The destination part should display the Paste As dialog box.

Result Types

The following types and constants represent the results of drag-and-drop operations.

`ODDragResult`

A result of type `ODBoolean` (page 848) indicating whether a drop is allowed in the specified facet.

`ODDropResult`

An enumeration specifying the result of a drop operation.

Constants of this type

`kODDropCopy`	Successful synchronous drop with copy semantics.
`kODDropFail`	Unsuccessful synchronous drop.
`kODDropMove`	Successful synchronous drop with move semantics.
`kODDropUnfinished`	An asynchronous drag-and-drop operation was started.

Asynchronous dragging is not currently supported on the Mac OS platform.

Linking

The following types and constants are used in linking operations.

`ODLinkInfo` A structure that contains information about a link destination for display in the Link Destination Info dialog box.

```
struct ODLinkInfo {
   ODType         kind;
   ODTime         creationTime;
   ODTime         changeTime;
   ODUpdateID     change;
   ODBoolean      autoUpdate;
};
```

Field descriptions

`kind` The part kind used by the link destination.

`creationTime` The date and time when this link destination was created.

`changeTime` The date and time of the latest source update read by this destination.

`change` The update ID of the latest source update read by this destination.

`autoUpdate` `kODTrue` if this destination updates automatically, otherwise `kODFalse`.

`ODLinkInfoAction`

An enumeration specifying the kind of action to be taken as the result of user selections in either the Link Source Info dialog box or the Link Destination Info dialog box.

Constants of this type

`kODLinkInfoBreakLink` Break the link.

`kODLinkInfoCancel` Take no action; the user canceled the dialog box.

`kODLinkInfoFindSource`
: Display the source of the link. Relevant only in the Link Destination Info dialog box.

`kODLinkInfoOk`
: Accept any new settings selected by the user in the dialog box.

`kODLinkInfoUpdateNow`
: Update the link immediately.

`ODLinkInfoResult`
: A structure that contains the results of user selections in either the Link Source Info dialog box or the Link Destination Info dialog box.

```
struct ODLinkInfoResult {
   ODLinkInfoAction  action;
   ODBoolean         autoUpdate;
};
```

Field descriptions

`action`
: The action taken by the user to dismiss the dialog box.

`autoUpdate`
: `kODTrue` if the user chose automatic updating, otherwise `kODFalse`.

`ODLinkKey`
: An opaque 32-bit type used to provide thread-safe access to link content. A link key is created when a link object or link-source object is locked; the link key must be used in any method that returns or modifies the content storage unit of the locked link object or link-source object.

Semantic Events and Scripting

This section describes the types and constants used in handling semantic events and scripting.

Application Shell

The following constant can be used in place of a destination part to send an Apple event to the application shell.

`kODAppShell` A pointer of type `ODPart*` that can be used to represent the application shell as a parameter to methods related to semantic events.

Apple Events

The following types are used for working with Apple events.

`ODDescType` A wrapper for the Apple events type `DescType`; an Apple events descriptor type. The section "Descriptor Types" on page 896 describes the constants defined for this type.

`ODEventClass` A wrapper for the Apple events type `AEEventClass`, the event class of an Apple event.

`ODEventID` A wrapper for the Apple events type `AEEventID`, the ID of an Apple event.

`ODSendMode` A wrapper for the Apple events type `AESendMode`, a context-specific mode associated with an Apple event.

`ODSendPriority` A wrapper for the Apple events type `AESendPriority`, a

priority specifying whether an Apple event is put at the back of the event queue or at the front of the queue.

Descriptor Types

This section describes the constants defined for the `ODDescType` type (page 895). You should use these constants only when writing an `'aete'` resource that overrides some of the standard scripting support provided by OpenDoc.

General Descriptor Types

The following descriptor types are used for a variety of purposes.

`kAEOpenDocSuite` The type code for the OpenDoc suite in the `'aete'` resource (value `'odst'`). You must use this suite code in your `'aete'` resource if you wish to inherit the OpenDoc suite from the system's `'aeut'` resources.

`kODStandardPartTokenType`
The type code for the standard part token, as returned from the name resolver's `Resolve` method (page 412); this type code is reserved and parts should never use it. This constant is equivalent to the `cPart` constant.

OpenDoc-Suite Classes

The following descriptor types correspond to the Apple events object model properties representing general elements (classes) of the OpenDoc suite.

`cDraft` A draft (value `'drft'`).

`cPart` A part (value `'part'`).

`cIconFamily` An icon family (value `'ifam'`).

Part-Information Properties

The following descriptor types correspond to the Apple events object model properties representing the properties shown in the Part Info dialog window.

`pASCreationDate`	The part's creation date (value `'ascd'`).
`pASModificationDate`	The part's modification date (value `'asmo'`).
`pAuthor`	The part's author (value `'auth'`).
`pBundled`	Specifies whether the part is bundled (value `'bndl'`).
`pCategory`	The part's category (value `'pcat'`).
`pComment`	A comment about the part (value `'comt'`).
`pContainer`	The part that contains this part (value `'ctnr'`).
`pEditor`	The part's editor (value `'edtr'`).
`pEditorName`	The part's name (value `'enam'`).
`pIcon`	The part's icon (value `'iimg'`).
`pKind`	The part kind of the part (value `'kind'`).
`pShowLinks`	Specifies that all part editors should display all link borders in all windows displaying the document (value `'slnk'`).
`pSize`	The part's size (value `'size'`).
`pStationery`	Specifies that the part is a stationery part (value `'stat'`).
`pUniqueID`	The ID number for the part (value `'ID  '`).
`pViewType`	The part's view type (value `'vwty'`).
`enumViewType`	The part's view type (value `'vwty'`). This constant is equivalent to the `pViewType` constant.

The section "View Types" on page 898 describes the descriptor types representing possible view types.

View Types

The following descriptor types represent possible view types for a part, as shown in the Part Info dialog window.

`kAEODFrame` Frame.

`kAEODLargeIcon` Large icon (standard icon).

`kAEODSmallIcon` Small icon.

`kAEODThumbnail` Thumbnail icon.

Extensions

The following constants of type `ODType` (page 846) represent various types for extensions to OpenDoc.

`kODExtSemanticInterface`
An extension to support a semantic interface in your part.

`kODSettingsExtension` An extension to add panels to the Part Info dialog box.

Name Spaces

The following type and constants are relevant to name spaces.

`ODNSTypeSpec`
An enumeration specifying the type of a name space.

Constants of this type

`kODNSDataTypeODObject`
An object name space.

`kODNSDataTypeODValue`
A value name space.

Binding

This section describes the types and constants used during the process of binding a part to a part editor.

Editors and Viewers

The following type and constant represent the shared libraries that implement OpenDoc part editors.

`ODEditor` An opaque, platform-specific type identifying a part editor.

Constants of this type

`kODNoEditor` A null OpenDoc editor.

The following constants of type `ODISOStr` (page 845) relate to part editors and part viewers.

`kODBlackBoxHandlerOfLastResort`
The editor ID for the editor of last resort, which is used for any part for which a suitable editor cannot be found.

`kODSimpleViewer` The viewer type for a simple part viewer.

Name-Mapping Resources

The following constants of type `ODISOStr` (page 845) identify the name-mapping (`'nmap'`) resources that a part editor uses to describe the kinds of

data it can edit. OpenDoc uses these resources to construct the specified tables. Constants marked [M] are specific to the Mac OS platform.

`kODCategoryUserString`
: A table that maps a part category to a user-visible string naming that category.

`kODContainerSuite`
: A table that maps a container type to the container suite ID for that type.

`kODEditorHelpFile`
: A table that maps an editor ID to the name of that editor's help file.

`kODEditorKinds`
: A table that maps an editor ID to the part kinds that the editor supports.

`kODEditorPlatformKind` [M]
: A table that maps an editor ID to the standard Mac OS kinds that the editor can edit. The table identifies each standard kind with four parts: a platform type space (either file or data), the Mac OS file type, a user-visible string naming the type, and the category that the type belongs to.

`kODEditorUserString`
: A table that maps an editor ID to a user-visible string naming that editor.

`kODKind`
: A table that maps a part kind to the categories that the kind belongs to.

`kODKindOldMacOSType` [M]
: A table that maps a new OpenDoc kind to the old Mac OS file type for that kind.

`kODKindUserString`
: A table that maps a part kind to a user-visible string naming that kind.

`kODViewer`
: A table that maps a part viewer ID to the viewer type. Currently, the only supported viewer type is `kODSimpleViewer`.

Part Categories

The following constants of the `ODISOStr` type (page 845) are used to identify part categories in the name-mapping (`'nmap'`) resources. A part category is a general classification of the format of data handled by a part editor.

`kODCategoryCalendar` Calendar data.

`kODCategoryChart` Chart data.

`kODCategoryCompressed`
Compressed data, such as in `.sit` or `.cpt` format.

`kODCategoryConnection`
Connection data, such as the CommToolBox Connection Tool preferences format.

`kODCategoryControl` Controls, such as a button.

`kODCategoryControlPanel`
Control panels, such as Editor Setup.

`kODCategoryDatabase` Databases, such as in FileMaker format.

`kODCategoryDrawing` Drawing data, such as in PICT or MacDraw format.

`kODCategoryExecutable`
Executable data, such as in `.COM`, `.EXE`, or other executable format.

`kODCategoryForm` Form data.

`kODCategoryFormula` Formula data, such as in an equation-editor format.

`kODCategoryKey` Key data, such as an AppleShare password or a PGP key.

`kODCategoryLocator` Locator data, such as a URL.

`kODCategoryMailingLabel`
Mailing labels, such as for a PowerTalk mailer.

`kODCategoryOutline` Document-outline data.

`kODCategoryMovie` Movie data, such as in QuickTime format.

`kODCategoryPageLayout`
Page-layout data.

`kODCategoryPainting` Painting data, such as in TIFF or MacPaint format.

`kODCategoryPersonalInfo`
Personal information, such as on business cards.

`kODCategoryPlainText` Plain text, such as in ASCII or Apple Standard Roman.

`kODCategoryPresentation`
Presentation data.

`kODCategoryPrinter` Printer data.

`kODCategoryProject` Project-management data, such as in MacProject format.

`kODCategoryQuery` Database queries, such as in SQL format.

`kODCategorySampledSound`
Sampled sounds, such as in the `'snd'` format.

`kODCategoryScript` Scripting data, such as in HyperTalk, AppleScript, or any other OSA-compliant scripting-language format.

`kODCategorySignature` Digital-signature data, such as a PowerTalk signature.

`kODCategorySpace` Space data, such as a folder, hard disk, or server.

`kODCategorySpreadsheet`
Spreadsheet data, such as in SYLK format.

`kODCategoryStructuredSound`
Structured sound, such as in MIDI format.

`kODCategoryStyledText`
: Styled text, such as in `'styl'` or MacWrite II format.

`kODCategory3DGraphic`
: 3D graphics data, such as in QuickTime VR format.

`kODCategoryTable`
: Tables, such as in tab-delimited text format.

`kODCategoryTime`
: Data related to time, for example, data used by clock parts.

`kODCategoryUtility`
: Data used by miscellaneous utilities.

Data Types in Resources

The following constants of type `ODISOStr` (page 845) identify the various data types used in the name-mapping (`'nmap'`) resources. Constants marked [M] are specific to the Mac OS platform.

`kODIsAnISOStringID`
: An ISO string.

`kODIsAnISOStringListID`
: A list of ISO strings.

`kODIsINTLTextID`
: International text structure.

`kODIsMacOSTypeID` [M]
: A file type specified with the Mac OS type `OSType`.

`kODIsPltfrmTypeSpacID`
: A platform type space (file or data).

`kODIsHelpFileNameID`
: A help file-name.

Error Codes

This section describes the constants defined for the `ODError` type (page 848).

The following error codes represent the exceptions that can be raised when you call an OpenDoc method. Error codes marked [M] are specific to the Mac OS platform

`kODErrAlreadyImportedLink` Failure to create a link due to an internal error.

`kODErrAlreadyNotified` An error occurred and the user has already been notified about it.

`kODErrBackgroundClipboardClear` Attempt to clear the clipboard in a background process.

`kODErrBrokenLink` Internal error; the link source disconnected from its destinations.

`kODErrBrokenLinkSource` The link has been broken at the source.

`kODErrCannotAcquireFrame` Failure to re-create the frame object from the specified storage unit.

`kODErrCannotAcquireLink` Failure to re-create the link source or link object from the specified storage unit or link-specification object.

`kODErrCannotAcquirePart` Failure to re-create the part object from the specified storage unit.

`kODErrCannotAddAction` Attempt to add an action to the undo stack while another undo or redo operation was in progress.

`kODErrCannotAddProperty` Unable to add the given property to a storage unit.

`kODErrCannotAllocateDragItem`
: The drag-and-drop object cannot allocate storage for the item to be dragged.

`kODErrCannotChangePermissions`
: Attempt to change permissions of a draft that has already been retrieved with different permissions.

`kODErrCannotCollapseDrafts` Attempt to collapse drafts specified by an invalid range of drafts.

`kODErrCannotCreateContainer`
: Failure to create a container because its specified type is not valid.

`kODErrCannotCreateFrame` Failure to create the requested frame.

`kODErrCannotCreateLink` Failure to create the requested link source or its companion link object.

`kODErrCannotCreatePart` Failure to create the requested part object.

`kODErrCannotCreateWindow` The window-state object cannot create a window.

`kODErrCannotEmbed` Attempt to access embedded frames for a part that does not support embedding.

`kODErrCannotEstablishLink` A persistent link could not be established.

`kODErrCannotFindLinkSource` Failure to locate the source of a cross-document link.

`kODErrCannotFindLinkSourceEdition` [M]
: Failure to locate the source of a cross-document link because the edition file does not exist.

`kODErrCannotGetExternalLink` [M]
: The link manager cannot create the specified cross-document link because the

	process that created the link specification is no longer running.
`kODErrCannotMarkAction`	Failure to start an action subhistory by placing a mark at the beginning of the undo and redo stacks; the undo object was never initialized properly.
`kODErrCannotOpenContainer`	Failure to open the physical container.
`kODErrCannotRegisterDependent`	A link object cannot register a dependent at this time.
`kODErrCannotRevealLink` [M]	The source of a cross-document link cannot be shown because the link has been broken at the source.
`kODErrCantCountFromLists`	Attempt to count items in a container that is not a list.
`kODErrCanvasHasNoOwner`	The specified canvas has no owning part.
`kODErrCanvasNotFound`	The specified canvas does not exist.
`kODErrCloningInProgress`	Attempt to begin a cloning transaction while another cloning transaction is in progress for the same draft.
`kODErrContainerDoesNotExist`	Failure to get a container object because no container exists with the specified identifier.
`kODErrContainerExists`	Failure to create a container object because a container with the specified identifier already exists.
`kODErrCorruptLink`	Internal error; a link object cannot be created from persistent storage because the data is not in a recognized format.

`kODErrCorruptLinkSource`	Internal error; a link-source object cannot be created from persistent storage because the data is not in a recognized format.
`kODErrCorruptLinkSpecValue`	The focused storage unit contains an invalid link-specification value.
`kODErrDocNotSaved` [M]	Internal error; the link manager could not create a cross-document link.
`kODErrDocumentDoesNotExist`	Failure to get a document object because no document exists with the specified identifier.
`kODErrDoesNotDrop`	This part does not support drag and drop.
`kODErrDoesNotLink`	This part does not support linking.
`kODErrDoesNotUndo`	This part does not support undo/redo.
`kODErrDraftDoesNotExist`	Failure to get a draft object because no draft exists with the specified identifier.
`kODErrDraftHasBeenDeleted`	Attempt to operate on a draft that has been deleted.
`kODErrDragItemNotFound`	The drag-and-drop object cannot find the item to be dragged.
`kODErrDragTrackingException`	An exception occurred in the systemwide mouse tracking service.
`kODErrEmptyStack`	The undo or redo stack is empty; the undo object was never initialized properly.
`kODErrFacetNotFound`	The requested facet was not found.
`kODErrFatalContainerError`	A fatal error occurred in the container suite. This error must be propagated.

`kODErrFocusAlreadyRegistered`
: The specified focus has already been registered.

`kODErrFocusNotRegistered`
: A requested focus is not registered.

`kODErrFrameHasFacets`
: The specified frame still has attached facets.

`kODErrIllegalClipboardCloneKind`
: One of the methods `ActionDone`, `ActionUndone`, or `ActionRedone` of the clipboard object is called with a clone kind other than `kODCloneCopy`, `kODCloneCut`, or `kODClonePaste`.

`kODErrIllegalNonTopmostDraft`
: The specified draft is not the topmost (most recent) draft of the document.

`kODErrIllegalNullContainerInput`
: A method was passed null for a container parameter that cannot be null.

`kODErrIllegalNullDispatchModuleInput`
: A dispatcher method was passed null for the dispatcher module.

`kODErrIllegalNullDocumentInput`
: A method was passed null for a document parameter that cannot be null.

`kODErrIllegalNullDraftInput`
: A method was passed null for a draft parameter that cannot be null.

`kODErrIllegalNullFacetInput`
: A frame method was passed null for a facet parameter that cannot be null.

`kODErrIllegalNullFrameInput`
A frame method was passed null for a frame parameter that cannot be null.

`kODErrIllegalNullIDInput` A method was passed null for an ID parameter that cannot be null.

`kODErrIllegalNullInput` A method was passed null for a parameter that cannot be null.

`kODErrIllegalNullPartInput` A frame method was passed null for a part parameter that cannot be null.

`kODErrIllegalNullPropertyInput`
A storage-unit method was passed null for a property parameter that cannot be null.

`kODErrIllegalNullShapeInput`
A frame method was passed null for a shape parameter that cannot be null.

`kODErrIllegalNullStorageSystemInput`
A frame method was passed null for the storage system parameter, which cannot be null.

`kODErrIllegalNullStorageUnitInput`
A method was passed null for a storage unit parameter that cannot be null.

`kODErrIllegalNullSUCursorInput`
A storage-unit method was passed null for a storage-unit cursor parameter that cannot be null.

`kODErrIllegalNullTokenInput`
A frame method was passed null for a token parameter that cannot be null.

`kODErrIllegalNullTransformInput`
A frame method was passed null for a transform parameter that cannot be null.

`kODErrIllegalNullValueTypeInput`
A frame method was passed null for a value type parameter that cannot be null.

`kODErrIllegalOperationOnSU` The requested operation cannot be performed on the specified storage unit.

`kODErrIllegalPropertyName` The specified property name is improperly formed or illegal.

`kODErrIllegalRecursiveEmbedding`
Attempt to embed a frame in its own part or in a containing part of its part.

`kODErrInconsistentCloneKind`
The specified clone kind is used inconsistently. For example, a paste or drop clone kind can occur only following a copy or cut operation.

`kODErrInsufficientInfoInParams`
The specified parameters to the method or function do not supply sufficient information, probably because all are null.

`kODErrInvalidBelowDraft` Attempt to create a draft after (above) a draft that is not the most recent (top) draft of the document.

`kODErrInvalidBlock` Attempt to access an invalid memory block, probably due to a bad heap or damaged object.

`kODErrInvalidCloneKind` The specified clone kind is not valid.

`kODErrInvalidCommandID` [M] No menu item exists with the specified command ID.

`kODErrInvalidDestinationDraft`
The specified destination draft is not valid for the specified clone kind.

`kODErrInvalidDraftID`	The specified draft ID is not valid in the context in which it is used.
`kODErrInvalidDraftKey`	The specified draft key is not the draft key for the current cloning transaction.
`kODErrInvalidExtension`	This extension object is invalid and should not be used because its base object no longer exists.
`kODErrInvalidFacet`	The specified facet is not valid in the context in which it is used; for example, it is not the child or parent of the facet whose method is being called.
`kODErrInvalidFrame`	The specified frame is not valid in the context in which it is used; for example, it is not a display frame of the part whose method is being called.
`kODErrInvalidGraphicsSystem`	This implementation of OpenDoc does not support the specified graphics system, that graphics system is not installed or available, or there is no drawing structure or print job associated with that graphics system.
`kODErrInvalidID`	The specified draft identifier is not valid in the context in which it is used.
`kODErrInvalidIterator`	The specified embedded-frames iterator is invalid and should not be used because the part that created it no longer exists.
`kODErrInvalidITextFormat`	The format of an `ODIText` structure (page 845) is not recognized.
`kODErrInvalidLinkKey`	The specified link key is not valid.
`kODErrInvalidLinkStatus`	The specified link status is not valid.

`kODErrInvalidNSName` Failure to read a stored name space because the stored name does not match that of the name space being read.

`kODErrInvalidNSType` The specified name space is of the wrong type (for example, it is a value name space but the method expects an object name space).

`kODErrInvalidObjectType` The specified object type is not valid in the context in which it is used.

`kODErrInvalidPermissions` The attempted action is not consistent with existing draft permissions.

`kODErrInvalidPersistentFormat`
Failure to read from persistent storage because the data is not in a recognized format.

`kODErrInvalidPersistentObject`
The specified persistent object is not valid.

`kODErrInvalidPersistentObjectID`
The specified identifier for a persistent object is not valid.

`kODErrInvalidPlatformShape` The requested operation cannot be performed on the specified platform shape.

`kODErrInvalidStorageUnit` The specified storage unit is not valid or the specified persistent object has no storage unit.

`kODErrInvalidStorageUnitKey`
Attempt to lock or unlock a storage unit with an invalid key.

`kODErrInvalidStorageUnitRef`
The specified persistent storage-unit reference is not valid.

`kODErrInvalidValueType`	The specified value type is improperly formed or illegal.
`kODErrIteratorNotInitialized`	A method was called on an uninitialized iterator.
`kODErrIteratorOutOfSync`	This iterator is invalid because its list was changed after the iterator was created.
`kODErrKeyAlreadyExists`	Failure to create a name-space object because a name space with the specified name already exists.
`kODErrLinkAlreadyExported` [M]	Internal error; the link manager cannot create a cross-document link because the specified link has already been exported.
`kODErrMoveIntoSelf`	A clone operation attempted to move a part into one of its embedded frames or embed one of the part's display frames into another of its display frames.
`kODErrNoBeginAction`	The undo object cannot find the begin action for this end action.
`kODErrNoDraftProperties`	Failure to create the storage unit to store draft properties.
`kODErrNoDragManager`	No platform-specific drag system service is available.
`kODErrNoDragSystemStorage`	The drag-and-drop object does not have system storage.
`kODErrNoEditionManager` [M]	The edition manager is not installed.
`kODErrNoLinkContent`	The link content storage unit has no contents property.

`kODErrNoLinkSpecValue`	The focused property does not contain a link-specification value.
`kODErrNonEmptyDraft`	Attempt to collapse nonempty drafts of a document.
`kODErrNoPreviousDraft`	Attempt to obtain the previous draft for the base draft of a document.
`kODErrNoPromises`	This part does not fulfill promises.
`kODErrNoShapeGeometry`	A shape that is being used as a polygon lacks geometric information (its polygonal representation).
`kODErrNoSysTranslationFacility`	No platform or system translation facility is installed. For example, this error will be thrown if Mac Easy Open is not installed on the Mac OS platform.
`kODErrNotAnODToken`	The specified token is not an OpenDoc token.
`kODErrNotExportedLink` [M]	Internal error; the specified link is not exported.
`kODErrNotImplemented`	The called method has not been implemented.
`kODErrNotImportedLink` [M]	Internal error; the link source is not imported.
`kODErrNotRootFrame`	This frame is not the root frame.
`kODErrNoValueAtThatIndex`	Failure to focus a storage unit because there is no value at the specified index.
`kODErrNullDestinationFrame`	The parameter specifying the destination frame is null.

`kODErrNullFacetInput`	A data-transfer method was passed null for a facet parameter that cannot be null.
`kODErrNullLinkInfoInput`	A data-transfer method was passed null for an `ODLinkInfo` parameter (page 893) that cannot be null.
`kODErrNullLinkInfoResultInput`	A data-transfer method was passed null for an `ODLinkInfoResult` parameter (page 894) that cannot be null.
`kODErrNullPasteAsResultInput`	A data-transfer method was passed null for a `ODPasteAsResult` parameter (page 888) that cannot be null.
`kODErrObjectNotInitialized`	An object has not been initialized properly.
`kODErrOutOfMemory`	Not enough memory to perform the specified operation (which involves an allocation).
`kODErrOutstandingDraft`	The attempted action would invalidate an outstanding draft (that is, one that is currently being referenced by some object).
`kODErrPartInUse`	The part for a specified wrapper is currently in use.
`kODErrPartNotWrapper`	A method was called on a part object instead of its encapsulating part wrapper.
`kODErrPropertyDoesNotExist`	The specified storage unit does not have the specified property.
`kODErrRefCountGreaterThanZero`	Attempt to delete a reference-counted object while it is being used (so its reference count is greater than 0).

`kODErrRefCountNotEqualOne` Attempt to delete a global reference-counted object while it is being used. (When a global object is not being used, only the session object has a reference to it, so its reference count is 1.)

`kODErrStorageUnitNotLocked` Attempt to unlock a storage unit that is not locked.

`kODErrSubClassResponsibility`
The called method should have been, but was not, overridden by the subclass of the class that defines the method.

`kODErrSUValueDoesNotExist` The specified storage-unit value does not exist.

`kODErrTransformErr` Attempt to perform an illegal operation on a transform object.

`kODErrUndefined` An undefined error occurred.

`kODErrUnfocusedStorageUnit` Attempt to perform an operation on a storage unit that is not properly focused.

`kODErrUnknownDragImageType` The drag-and-drop object does not recognize the specified drag-image type.

`kODErrUnknownExtension` The specified extension is not a known extension.

`kODErrUnknownLinkSpecVersion` [M]
The version of the link specifier is unknown.

`kODErrUnknownUpdateID` The specified update ID is the reserved value `kODUnknownUpdate`.

`kODErrUnsupportedExtension` Failure to get an object's extension because the object does not support the specified extension.

`kODErrUnsupportedFramePositionCode`	The frame position code specified for a new facet is not recognized.
`kODErrUnsupportedPosCode`	The specified position code is not supported in the context in which it is used.
`kODErrValueIndexOutOfRange`	The specified property has no value at the specified index.
`kODErrValueOutOfRange`	A numeric value does not fall within the correct range of values.
`kODErrZeroRefCount`	Attempt to decrement an object's reference count that is already 0.

The following additional constants of type `ODError` (page 848) do not represent error conditions.

`kODMaxError`	The maximum numeric value for OpenDoc error codes.
`kODMinError`	The minimum numeric value for error OpenDoc codes.
`kODMinUsedError`	The minimum numeric value for error codes currently in use.
`kODNoError`	No error occurred.

Shell Plug-In Installation Function

On the Mac OS platform, a **shell plug-in** is an import library that extends the capabilities of the document shell or adds session-wide functionality to OpenDoc. An OpenDoc document can use a particular shell plug-in only if the import library is located in the OpenDoc Shell Plug-Ins folder on the user's machine when the document is opened. This appendix describes requirements for runtime installation of a shell plug-in on the Mac OS.

An import library is a shared library that is automatically loaded at runtime by the Code Fragment Manager. If you implement a shell plug-in, it must follow the conventions for import libraries; it should have the file type `'shlb'`. Your shell plug-in must have an **installation function** that is an exported entry point. The installation function should install whatever functionality the shell plug-in needs. As is the case for all import libraries, a shell plug-in can optionally have an initialization function and a termination function that are exported as entry points.

When a document is opened, OpenDoc installs all available shell plug-ins. To install a shell plug-in, OpenDoc creates a connection to the import library and executes the shell plug-in's installation function. Specifically, OpenDoc gets a pointer to the `ODShellPluginInstall` function (page 922) from the import library, casts that pointer to the `ODShellPluginInstallProc` type (page 921), and calls the function. If installation is successful, OpenDoc then performs any actions requested by the installation function; for example, the function might request OpenDoc to close its connection to the import library. The function uses a value of type `ODShellPluginActionCodes` (page 920) to specify the desired actions.

For more information about the operation of shell plug-ins and their appropriate location within OpenDoc folders, see the chapter on extending OpenDoc in the *OpenDoc Programmer's Guide for the Mac OS*. For information about import libraries, see the chapter on the Code Fragment Manager in *Inside Macintosh: PowerPC System Software*.

Types and Constants

This section describes the types and constants used by shell plug-in installation functions.

`ODShellPluginActionCodes`

An unsigned 32-bit value used to represent a collection of action codes. Each action code is a flag corresponding to a particular action. If a flag is set, the specified action should be performed; otherwise, the action should not be performed.

Constants of this type

`kODShellPluginCloseConnection`

The document shell should close its connection to the shell plug-in import library. If no other code fragment has a connection to the import library, the Code Fragment Manager unloads the library, releasing the memory occupied by its code and data.

`kODShellPluginNoAction`

No actions should be performed.

Because these constants are `#define` constants, no memory is allocated for them; as a result, you should not attempt to obtain a pointer to any of them. For example, the following statement is illegal:

```
//    Illegal attempt to take the address of
//    a #define constant!
action = *kODShellPluginCloseConnection;
```

`ODShellPluginInstallProc`

A pointer to a shell plug-in installation function.

```
#ifdef __cplusplus
extern "C" {
#endif

typedef OSErr (*ODShellPluginInstallProc)(
                    Environment*,
                    ODDraft*,
                    ODShellPluginActionCodes*);

#ifdef __cplusplus
}
#endif
```

A value of type `ODShellPluginInstallProc` is a pointer to a function that follows C calling conventions. It takes three parameters: a pointer to a SOM environment parameter, a pointer to an object of the `ODDraft` class (page 145), and a pointer to a value of type `ODShellPluginActionCodes` (page 920). The function returns an error code of type `OSErr`.

Programmer-Defined Functions

The programmer who implements a shell plug-in must define its installation function. When you implement the installation function for your shell plug-in, you must be careful to observe the following conventions:

- Your function must be named `ODShellPluginInstall`; case is important. If you misspell the name, your shell plug-in will not be installed.
- Your function's parameters and return type must match those specified for the `ODShellPluginInstallProc` type.
- If installation succeeds, your function must return the value `noErr`.
- The compiler must export the name of your function as an entry point.

- Your function must follow C calling conventions. If you use C++, you must surround your installation function's declaration with an `extern "C"` declaration to ensure that parameters are passed in the correct order on 680x0 systems.

The following description of the `ODShellPluginInstall` function illustrates the form of a shell-plug-in installation function.

ODShellPluginInstall

Mac OS

The `ODShellPluginInstall` function should install whatever functionality your shell plug-in requires.

```
OSErr ODShellPluginInstall (
                        Environment* ev,
                        ODDraft* draft,
                        ODShellPluginActionCodes* action);
```

`ev`	A pointer to the SOM environment parameter used for passing exceptions.
`draft`	A pointer to the most recent draft of the document being opened.
`action`	A pointer to the value in which to return action codes specifying actions OpenDoc should perform when this function returns successfully.
return value	`noErr` if the installation is successful, otherwise a Mac OS error code indicating why installation failed.

DISCUSSION

When the user opens a document, OpenDoc installs all shell plug-in import libraries located in the OpenDoc Shell Plug-Ins folder on the user's machine. To install a shell plug-in, OpenDoc creates a connection to the import library and calls the `ODShellPluginInstall` function of each import library. If no other code fragment already has a connection to the shell plug-in, the Code Fragment Manager loads the shell plug-in into memory.

The `draft` parameter points to the draft object for the current draft of the document.

When this function is called, the `action` parameter points to a value that is equal to `kODShellPluginNoAction`. If you want OpenDoc to perform particular actions following installation of your shell plug-in, you should set the appropriate flags in that value. You can set the flag for a particular action using a bitwise `OR` operator (for example, the `|` or `|=` operator in C++). Currently, the only supported action is closing the connection to the shell plug-in import library (`kODShellPluginCloseConnection`).

IMPORTANT

To request actions, you must modify the value that the `action` parameter addresses (`*action`). Do not modify the value of the `action` parameter itself. ▲

If the installation is successful, this function should return `noErr`. In that case, the document shell performs any actions specified in the value to which the `action` parameter points. The shell plug-in is then available for the document to use.

If the installation fails for any reason, this function should return an error code. In that case, OpenDoc displays a dialog box asking the user to remove the shell plug-in from the system.

The following code fragment illustrates how to declare and define the C or C++ installation function for your shell plug-in.

```
//    If you use C++, declare your function within an
//    extern "C" declaration.
#ifdef __cplusplus
extern "C" {
#endif

OSErr ODShellPluginInstall ( Environment* ev,
                        ODDraft* draft,
                        ODShellPluginActionCodes* action);

#ifdef __cplusplus
}
#endif
```

```
//    Define your function.
OSErr ODShellPluginInstall(Environment* ev,
                           ODDraft* draft
                           ODShellPluginActionCodes* action);
{
   //    Perform any necessary installation; return an
   //    error code if installation fails.
   ...
   //    If appropriate, set action codes in *action.
   *action |= kODShellPluginCloseConnection;
   //    Return noErr if installation succeeds.
   return noErr;
}
```

SEE ALSO

The `ODShellPluginActionCodes` type (page 920).
The `ODDraft` class (page 145).

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Y

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Proof pages and final pages were created on an Apple LaserWriter Pro printer. PostScript™, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

WRITERS
Denise Stone, Carli Scott, Stan Kelly-Bootle, John Wendt, Bob Morrish, Ulla Hald

DEVELOPMENTAL EDITORS
Wendy Krafft, Antonio Padial

PRODUCTION EDITORS
Alexandra Solinski, Lorraine Findlay

ELECTRONIC MEDIA TEAM
Dan Peterson, Liz Hujsak, Bill Harris

LEAD WRITERS
Dave Bice, Alan Spragens

Special thanks to Jens Alfke, Craig Carper, Tantek Çelik, Caia Grisar, Vincent Lo, Nick Pilch, Richard Rodseth, Joshua Susser

Acknowledgments to Troy Gaul, Mike Halpin, Doug Hill, Eric House, Barbara Kozlowski, Michael Mazour, David McCusker, Jon Pugh, Thomas Weisbach