ALOE API

1.0 Draft
February 4, 1997

 Apple Computer, Inc.

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

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

I. ABOUT THIS DOCUMENT

The purpose of this document is to introduce ALOE, the Apple Library for Object Ebedding. ALOE is a companion technology to OpenDoc. It enables developers to add OpenDoc container support to traditional applications, both new and old. This is achieved by encapsulating the OpenDoc APIs behind a simpler API focused on the issues surrounding container support.

Section I provides an overview of the ALOE product. Section II examines the requirements and the initial feature set of ALOE. Section III presents the deliverable components of ALOE. Section IV and V discusses the architecture and API.

Before proceeding, you should know that:

1) This is a working document. Any information contained here may be changed in the future.

2) This is a draft. It may contain incomplete information and mistakes. However, we have tried to avoid those as much as we could. Documentation written and edited by professional writers will come in the near future.

3) Your input is extremely valuable to us. Please let us know if you have any comments or questions.

II. REQUIREMENTS AND FEATURE SET

Requirements

Functionality

Preserve existing application functionality

An application does not need to compromise its existing functionality because of adoption of ALOE and OpenDoc.

Provide sufficient OpenDoc functionality

Even though ALOE is not designed to expose all functionality of OpenDoc, it should provide a sufficient set of features so that it is a viable solution to most applications.

Programmability

Simple To Use

The API of ALOE must be simple to use and integrate into an existing application. The learning curve for ALOE API should be as flat as possible.

Non-intrusive

In order to support a new technology, an application inevitably has to be changed. ALOE needs to be as non-intrusive as possible. Application developers need to feel that they are not changing their application's architecture to accommodate OpenDoc.

Language Support

The API of ALOE must be accessible in multiple languages. The first language to be supported will be C (and hence C++). Interfaces can be created for other languages as needed.

Compatibility

Operating System Compatibility

ALOE needs to run on all systems on which OpenDoc runs.

System configuration

Many container applications are going to be running both on systems with OpenDoc and ALOE and on systems without. An application which has incorporated ALOE should be able to function on all these systems.

Graphics Systems

ALOE needs to support the main graphics systems on each platform. On the Mac OS, both Classic QuickDraw and QuickDraw GX are to be supported. Choice of graphics system should not be an impediment to ALOE adoption.

Compatibility with OpenDoc part editors

No OpenDoc part editors should need to be changed in order to run with ALOE. In other words, a container application can take advantage of the full spectrum of part editors available.

Performance

Footprint

ALOE and OpenDoc should only be loaded when the functionality is required. Therefore, the container application should see very little in its footprint if ALOE and OpenDoc functionality is not triggered.

In addition, the application should have full control when to load and unload ALOE. This enables applications to optimize performance in a custom way.

Interacting with OpenDoc

ALOE should not impose significant overhead on the normal execution of the application. ALOE should have well-defined and predictable performance behaviors.

Initial Feature Set

The following is a list of features provide by the first release of ALOE:

- Initialization and shutdown of OpenDoc
- Event handling of OpenDoc-related events
- Management of embedded OpenDoc parts
- Persistent storage management for embedded OpenDoc parts
- Window management of OpenDoc and hybrid windows
- Support for application's floating window scheme
- Menu handling
- Rendering of embedded OpenDoc content (both on-screen and off-screen)
- Support for asynchronous drawing by the application
- Complete support for Clipboard, Drag-and-Drop and Linking of embedded content to embedded content.

- Complete support for scripting
- Mouse region calculation / Sleep time calculation / Event mask setup
- Modal state support (both on the application and in OpenDoc)
- Reconcile undo mechanisms between OpenDoc and the application
- Support for various states for embedded part defined by OpenDoc
- Direct activation of embedded OpenDoc part
- Notification of change in content for editions

III. DELIVERABLES

To developers, ALOE is a set of APIs which enables them to add OpenDoc support to their applications. The development kit includes:

- Header files for ALOE (ALOE.h, ALOE.rsrc).
- Proxy root part editor (debug and non-debug versions) to be installed in the Editors folder.
- Static library (debug and non-debug versions) to be linked with the container application.
- Shared library (debug and non-debug versions) of ALOE.
- Sample application and associated sample code.
- Developer documentation.

To end users, ALOE is delivered as a shared library (which is the non-debug version of the Shared Library mentioned above) and a part editor (which is the proxy root part editor mentioned above). It will be installed with OpenDoc or the application which uses it.

IV. ARCHITECTURE OVERVIEW

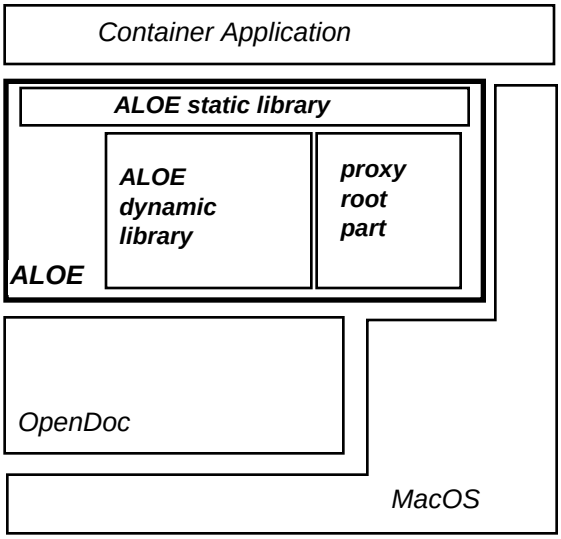
This section describes the overall architecture of ALOE.

ALOE consists of the following pieces:

1) ALOE static library

2) ALOE shared (dynamic) library

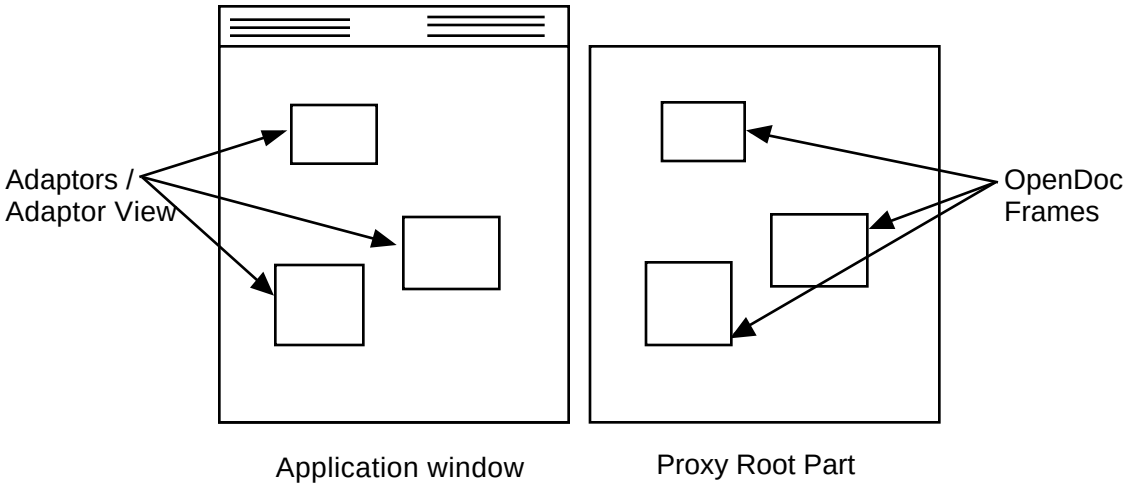
3) Proxy root part



An application links against the ALOE static library, which provides some default behavior when the shared library or OpenDoc is not installed. The static library is also responsible for dynamically loading the ALOE shared library when the container application wants to access ALOE/OpenDoc functionality.

The ALOE shared library provides most of the functionality. It acts as an interface between the container application and OpenDoc. It mediates access to shared resources (e.g., windows and menus) between the application and OpenDoc so that they do not interfere with each other.

ALOE shared library uses the proxy root part to represent the application document. The proxy root part is needed because parts need a place to be embedded in. Also, many of the OpenDoc recipes require the existence of a root part. This proxy root part is a very simple containing part which handles basic layout and imaging protocols.



In ALOE, there are several objects for embedding:

1) Adaptor

   An Adaptor corresponds to an OpenDoc part editor. It represents an opaque embedded entity.

2) Adaptor View

   In OpenDoc terms, an Adaptor View is the embodiment of a frame and a facet.

   An Adaptor View handles the layout and rendering of an Adaptor. In general, each Adaptor has only one Adaptor View. Whenever the application wants to change the Adaptor's position or draw the Adaptor, it will call the Adaptor's View to do it.

   Multiple views provide utility and convenience to certain applications. A typical example is when the application splits its document window. In this case, the application may create two Views on the same Adaptor, each View in one split pane.

3) Adaptor Manager

   An Adaptor Manager is associated with each application document. The AdaptorManager is more storage-oriented than display-oriented. Even if a document has multiple windows, only one Adaptor Manager should be created. However, if there are two documents in the same process, two Adaptor Managers are needed.

   The main purposes of the Adaptor Manager is to deal with storage and manage the Adaptors.

V. EXTERNAL FUNCTIONAL/INTERFACE SPECIFICATION

This section describes the interface of ALOE. The first subsection lists the API by components. The second subsection lists the API in terms of application requirements. The third subsection provides detailed description of each function.

By Components

ALOEAdaptorMgr

Creation and Destruction

```
OSErr ALOENewAdaptorMgr(ALOEAdaptorMgr* newAdaptorMgr);

OSErr ALOENewAdaptorMgrFromFile(long refNum,
                        ALOEAdaptorMgr* newAdaptorMgr);

OSErr ALOENewAdaptorMgrFromMemory(Handle source,
                        ALOEAdaptorMgr* newAdaptorMgr);

OSErr ALOEDisposeAdaptorMgr(ALOEAdaptorMgr adaptorMgr);
```

Storage

```
OSErr ALOEIsAdaptorMgrDirty(ALOEAdaptorMgr adaptorMgr,
                        Boolean* changed,
                        Boolean* isSameSize);

OSErr ALOEWriteAdaptorMgrToFile(ALOEAdaptorMgr adaptorMgr,
                        long refNum,
                        long* bytesWritten);

OSErr ALOENewHandleFromAdaptorMgr(
                        ALOEAdaptorMgr adaptorMgr,
                        Handle* destination,
                        Boolean allocateInTempMem);
```

ALOEAdaptor

Creation and Destruction

```
OSErr ALOENewAdaptor(ALOEAdaptorMgr adaptorMgr,
                        WindowPtr window,
                        ALOEISOString kind,
                        Ptr dataPtr,
                        long dataSize,
                        RgnHandle shape,
                        ALOEAdaptor* newAdaptor);

OSErr ALOENewSynchronizedAdaptor(ALOEAdaptor fromAdaptor,
                        WindowPtr window,
                        ALOEAdaptor* newAdaptor);
```



```
OSErr ALOEBeginDuplicateClone(
                        ALOEAdaptorMgr fromAdaptorMgr,
                        ALOEAdaptorMgr toAdaptorMgr,
                        WindowPtr window);

OSErr ALOEDuplicateAdaptor(
                        ALOEAdaptor fromAdaptor,
                        ALOEAdaptor* newAdaptor);

OSErr ALOEDisposeAdaptor(ALOEAdaptor adaptor);
```

Storage

```
OSErr ALOEGetPersistentID(ALOEAdaptor adaptor,
                        ALOEPersistentID id);

OSErr ALOEGetAdaptor(ALOEAdaptorMgr adaptorMgr,
                        ALOEPersistentID id,
                        WindowPtr window,
                        ALOEAdaptor* newAdaptor);
```

Activation

```
OSErr ALOESetActivateApplicationCallback(
                        ALOEActivateApplicationUPP proc,
                        void *contextPtr);

OSErr ALOEActivateAdaptor(ALOEAdaptor adaptor);

OSErr ALOESetAdjustActiveBorderShapeCallback(
                  ALOEAdjustActiveBorderShapeUPP proc,
                  void *contextPtr);

OSErr ALOESetDoesPropagateEvents(ALOEAdaptor adaptor,
                        Boolean doesPropagate);
```

Layout

```
OSErr ALOESetAdaptorShapeRequestCallback(
                        ALOEAdaptorShapeRequestUPP proc,
                        void *contextPtr);

OSErr ALOEGetAdaptorShape(ALOEAdaptor adaptor,
                        RgnHandle adaptorShape);

OSErr ALOESetAdaptorShape(ALOEAdaptor adaptor,
                        RgnHandle adaptorShape);
```

OpenDoc States

```
OSErr ALOESetAdaptorRemovedState(ALOEAdaptor adaptor,
                        Boolean removed);

Boolean ALOEGetAdaptorBundleState(ALOEAdaptor adaptor);
```



```
OSErr ALOESetAdaptorBundleState(ALOEAdaptor adaptor,
                        Boolean bundled);
```

Linking Support

```
OSErr ALOESetRevealAdaptorCallback(
                        ALOERevealAdaptorUPP proc,
                        void *contextPtr);

OSErr ALOESetAdaptorChangedCallback(
                        ALOEAdaptorChangedUPP proc,
                        void *contextPtr);
```

Utilities

```
ALOEAdaptorMgr ALOEGetAdaptorMgrFromAdaptor(
                        ALOEAdaptor adaptor);
```

ALOEAdaptorView

Creation and Destruction

```
OSErr ALOENewAdaptorView(ALOEAdaptor adaptor,
                        GrafPtr port,
                        Point offset,
                        THPrint printJob,
                        ALOEAdaptorView* newView);

OSErr ALOEDisposeAdaptorView(ALOEAdaptorView view);
```

Imaging

```
OSErr ALOEGetAdaptorViewOffset(ALOEAdaptorView view,
                        Point* offset);

OSErr ALOESetAdaptorViewOffset(ALOEAdaptorView view,
                        Point offset);

OSErr ALOEDrawAdaptorView(ALOEAdaptorView view,
                        RgnHandle viewClip);

OSErr ALOESetAdaptorViewScaleFactor(ALOEAdaptorView view,
                        Fixed horzScaleFactor,
                        Fixed vertScaleFactor);

OSErr ALOESetAdaptorImageUpdatedCallback(
                        ALOEAdaptorImageUpdatedUPP proc,
                        void *contextPtr);

OSErr ALOEGetAdaptorViewPicture(ALOEAdaptorView view,
                        PicHandle* image);
```

OpenDoc States

```
Boolean ALOEGetAdaptorViewSelectState(
                        ALOEAdaptorView view);
```



```
OSErr ALOESetAdaptorViewSelectState(ALOEAdaptorView view,
                        Boolean selected);

ALOEHighlightState ALOEGetAdaptorViewHighlightState(
                        ALOEAdaptorView view);


OSErr ALOESetAdaptorViewHighlightState(
                        ALOEAdaptorView view,
                        ALOEHighlightState highlighted);
```

Info Dialog

```
OSErr ALOEShowAdaptorInfo(ALOEAdaptorView view);
```

Utilities

```
ALOEAdaptor ALOEGetAdaptorFromAdaptorView(
                        ALOEAdaptorView view);
```

ALOE

Initialization and termination

```
OSErr StartALOE();

OSErr TerminateALOE();

OSErr ALOESetScratchVRefNum(short volRefNum);
```

Window Management

```
Boolean IsALOEWindow(WindowPtr window);

Boolean IsALOEFloatingWindow(WindowPtr window);

OSErr ALOESizeWindow(WindowPtr window,
                        short w,
                        short h,
                        Boolean fUpdate);

OSErr ALOEDragWindow(WindowPtr window,
                        Point startPt,
                        const Rect* boundsRect);

OSErr ALOESetWindowCallback(
                  ALOEWindowCallbackUPP windowCallbackProc,
                  void * contextPtr);

OSErr ALOEWindowActionDispatch(WindowPtr window,
                        ALOEWindowAction action,
                        Boolean param);
```



```
OSErr ALOEDeactivateWindow(WindowPtr window);
```

Window Utilities

```
OSErr ALOEAddWindow(WindowPtr window);

OSErr ALOERemoveWindow(WindowPtr window);

OSErr ALOEShowWindow(WindowPtr window);

OSErr ALOEHideWindow(WindowPtr window);

OSErr ALOESelectWindow(WindowPtr window);
```

Menu bar Management

```
OSErr ALOESetBaseMenuBar(short mbarResource);

OSErr ALOESetBaseMenuBarHandle(Handle mbarHandle);

OSErr ALOERegisterMenuCommand(short menuID,
                        short menuItem,
                        long commandID);

OSErr ALOEGetMenuCommand(short menuID,
                        short menuItem,
                        long* commandID);

OSErr ALOEAdjustMenus();

OSErr ALOESetResetMenuBarCallback(
                  ALOEResetMenuBarUPP resetMenuBarProc,
                  void * contextPtr);
```

Event handling

```
Boolean ALOEHandleEvent(EventRecord* event);

RgnHandle ALOEGetSleepRegion(RgnHandle appSleepRegion);

EventMask ALOEGetEventMask(EventMask appMask);

UInt32 ALOEGetSleepTime(UInt32 appSleepTime);

Boolean ALOEIdle();
```

Modal state handling

```
OSErr ALOEApplicationModal(Boolean requestModalFocus);

OSErr ALOEApplicationActive();
```

Undo Management


```
Boolean ALOEIsUndoClear();

OSErr ALOEClearUndo();
```

Clipboard

```
PScrapStuff ALOEInfoScrap();

short ALOEGetScrapCount();

Boolean ALOEScrapCountStillValid(short scrapCount);

Boolean ALOEIsScrapEmpty();

long ALOEZeroScrap();

long ALOEPutScrap(long length,
                  ResType theType,
                  Ptr source);

long ALOEGetScrap(Handle hDest,
                  ResType theType,
                  long *offset);

long ALOEUnloadScrap();

long ALOELoadScrap();

OSErr ALOEUpdateScrap();

OSErr ALOESetWriteScrapCallback(ALOEWriteScrapUPP proc,
                        void *contextPtr);

OSErr ALOEBeginScrapClone(ALOEAdaptorMgr adaptorMgr,
                        WindowPtr window,
                        ALOECloneKind kind);

OSErr ALOECloneAdaptorFromContent(ALOEAdaptor adaptor,
                        Point offset,
                        ALOEClonedID clonedId);

OSErr ALOECloneAdaptorToContent(ALOEClonedID id,
                        ALOEAdaptor* newAdaptor);

OSErr ALOEAbortClone();

OSErr ALOEEndClone();

OSErr ALOEEmbedScrap(ALOEAdaptorMgr adaptorMgr,
                        WindowPtr window,
                        ALOEAdaptor* newAdaptor);
```

Drag and Drop

```
OSErr ALOEBeginDragClone(ALOEAdaptorMgr adaptorMgr,
                        WindowPtr window,
```



```
                                DragReference theDragRef,
                                ALOECloneKind kind);

        OSErr ALOEAddDragItemFlavor(DragReference theDragRef,
                                ItemReference theItemRef,
                                FlavorType theFlavorType,
                                void* data,
                                Size dataSize,
                                FlavorFlags theFlags);

        Boolean ALOEAdaptorViewCanAcceptDrop(ALOEAdaptorView view);

        OSErr ALOEEmbedDrag(ALOEAdaptorMgr adaptorMgr,
                                WindowPtr window,
                                DragReference theDragRef,
                                long index,
                                ALOEAdaptor* newAdaptor);
```

Scripting

```
        OSErr ALOEObjectInit();

        OSErr ALOEInstallEventHandler(
                        AEEventClass theAEEventClass,
                        AEEventID theAEEventID,
                        AEEventHandlerUPP handler,
                        long handlerRefCon,
                        Boolean isSysHandler);

        OSErr ALOEGetEventHandler(
                        AEEventClass theAEEventClass,
                        AEEventID theAEEventID,
                        AEEventHandlerUPP* handler,
                        long* handlerRefCon,
                        Boolean isSysHandler);

        OSErr ALOERemoveEventHandler(
                        AEEventClass theAEEventClass,
                        AEEventID theAEEventID,
                        AEEventHandlerUPP handler,
                        Boolean isSysHandler);

        OSErr ALOEInstallPreDispatchHandler(
                        AEEventHandlerUPP handler
                        Boolean isSysHandler);

        OSErr ALOEGetPreDispatchHandler(
                        AEEventHandlerUPP* handler
                        Boolean isSysHandler);

        OSErr ALOERemovePreDispatchHandler(
                        AEEventHandlerUPP handler
                        Boolean isSysHandler);

        OSErr ALOEInstallObjectAccessor(
                        DescType desiredClass,
                        DescType containerType,
```



```
                    OSLAccessorUPP theAccessor,
                    long accessorRefCon,
                    Boolean isSysHandler);

OSErr ALOEGetObjectAccessor(
                    DescType desiredClass,
                    DescType containerType,
                    OSLAccessorUPP* theAccessor,
                    long* accessorRefCon,
                    Boolean isSysHandler);

OSErr ALOERemoveObjectAccessor(
                    DescType desiredClass,
                    DescType containerType,
                    OSLAccessorUPP theAccessor,
                    Boolean isSysHandler);

OSErr ALOESetObjectCallbacks(
                    OSLCompareUPP compareProc,
                    OSLCountUPP countProc,
                    OSLDisposeTokenUPP disposeTokenProc,
                    OSLGetMarkTokenUPP getMarkTokenProc,
                    OSLMarkUPP markProc,
                    OSLAdjustMarksUPP adjustMarksProc,
                    OSLGetErrDescUPP getErrDescProc);

OSErr ALOEInstallCoercionHandler(
                    DescType fromType,
                    DescType toType,
                    AECoercionHandlerUPP handler,
                    long handlerRefCon,
                    Boolean fromTypeIsDesc,
                    Boolean isSysHandler);

OSErr ALOEGetCoercionHandler(
                    DescType fromType,
                    DescType toType,
                    AECoercionHandlerUPP* handler,
                    long* handlerRefCon,
                    Boolean* fromTypeIsDesc,
                    Boolean isSysHandler);

OSErr ALOERemoveCoercionHandler(
                    DescType fromType,
                    DescType toType,
                    AECoercionHandlerUPP handler,
                    Boolean isSysHandler);

OSErr ALOEGetCoercionHandler(
                    DescType fromType,
                    DescType toType,
                    AECoercionHandlerUPP* handler,
                    long* handlerRefCon,
                    Boolean* fromTypeIsDesc,
                    Boolean isSysHandler);

OSErr ALOEResolve(AEDesc* objectSpecifier,
                    short callbackFlags,
```



```
                                AEDesc* theToken,
                                ALOEAdaptorView* tokenContext);

        OSErr ALOEDisposeToken(AEDesc* theToken);
```

Edition Manager Support

```
        Boolean ALOEReserveSectionID(long sectionID);
```

Utilities

```
        ALOEISOString ALOEGetKindFromResType(ResType type);
```


Scalable API

This subsection presents the API in a scalable fashion. It first lists the required API which every application must support. Then it lists the optional API. Developers may choose to adopt these if they fit their needs.

The following table shows the API calls which are required for all applications:

Required for all applications	`StartALOE`
	`TerminateALOE`
	`ALOESetActivateApplicationCallback`
	`ALOESelectWindow`
	`ALOESetBaseMenuBar`
	`ALOEHandleEvent`
	`ALOEGetSleepRegion`
	`ALOEGetEventMask`
	`ALOEGetSleepTime`

The following tables shows the supported features and their related APIs.

1) If your application does not support a certain feature, you do not have to use its related API calls. For an example, if your application supports offscreen drawing, you would need to use `ALOESetAdaptorImageUpdatedCallback`. If your application does not support the Edition Manager, you can ignore `ALOESetAdaptorChangedCallback`.

2) For certain features, there are both required and optional API calls.

3) The features are not presented in any special ordering.

Embedding	ALOENewAdaptorMgr ALOEDisposeAdaptorMgr ALOENewAdaptor ALOENewAdaptorView ALOESetAdaptorRemoved ALOEDisposeAdaptorView ALOEDrawAdaptorView ALOESetAdaptorViewSelectState ALOESetAdaptorViewHighlightState ALOESetAdaptorBundleState ALOESetRevealAdaptorViewCallback ALOEAdjustMenus ALOEShowAdaptorInfo ALOEApplicationModal ALOEApplicationActive
Embedding (optional)	ALOEBeginDuplicateClone ALOEDuplicateAdaptor ALOENewSynchronizedAdaptor ALOESetAdaptorShapeRequestCallback ALOEGetAdaptorShape ALOESetAdaptorShape ALOESetAdaptorRemovedState ALOESetAdaptorViewOffset ALOEGetAdaptorViewOffset ALOEGetAdaptorViewSelectState ALOEGetAdaptorViewHighlightState ALOEGetAdaptorBundleState ALOEGetAdaptorMgrFromAdaptor ALOEGetAdaptorFromAdaptorView ALOEGetKindFromResType
Persistent storage	ALOEIsAdaptorMgrDirty ALOEWriteAdaptorMgrStorageToFile ALOENewAdaptorMgrFromFile ALOENewHandleFromAdaptorMgr ALOENewAdaptorMgrFromMemory ALOEGetPersistentID ALOEGetAdaptor
Undo	ALOEIsUndoClear ALOEClearUndo
Clipboard	ALOEBeginScrapClone ALOECloneAdaptorToContent ALOECloneAdaptorFromContent ALOEAbortClone ALOEEndClone ALOEEmbedScrap ALOEInfoScrap ALOEZeroScrap ALOEPutScrap ALOEGetScrap ALOEUnloadScrap ALOELoadScrap ALOEIsScrapEmpty
Clipboard (optional)	ALOEGetScrapCount ALOEScrapCountStillValid ALOEUpdateScrap ALOESetWriteScrapCallback

Initiating drags	`ALOEBeginDragClone` `ALOEAddDragItemFlavor`
Receiving drops	`ALOEEmbedDrag`
Edition Manager support Enable applications supporting Edition Manager to continue to work	`ALOESetAdaptorChangedCallback` `ALOEReserveSectionID`
Cached Presentation Cached presentation refers to a static image of the part. This static image can be used when OpenDoc or ALOE is not available. This feature is not required on every application, but a developer may choose to support this for better user experience.	`ALOEGetAdaptorViewPicture`
Direct activation support Directly activating an adaptor is not required as ALOE handles adaptor activation or deactivation for the application.	`ALOEActivateAdaptor`
OpenDoc active border support ALOE provides default implementation for active border calculation. However, developers are encouraged to supply their own especially when their applications support overlapping shapes.	`ALOESetAdjustActiveBorderShapeCallback`
Offscreen drawing support Applications are in control of imaging. ALOE provides the following function for application which takes advantage of offscreen drawing.	`ALOESetAdaptorImageUpdatedCallback`

Floating window support
 	
ALOE handles normal window management. However, if an application has its own floating window scheme, ALOE provides functions to reconcile the two schemes.
	`IsALOEWindow`
	`IsALOEFloatingWindow`
	`ALOESetWindowCallback`
	`ALOEDeactivateWindow`

Menu command support
 	
Both OpenDoc and ALOE have adopted the menu command schemes. If your application supports the same scheme, you can use the ALOE functions for convenience.
	`ALOESetBaseMenuBarHandle`
	`ALOERegisterMenuCommand`
	`ALOEGetMenuCommand`

Scripting support
 	
If an application supports scripting, it should use the ALOE functions to manage the Apple Event Manager functions.
	`ALOEObjectInit`
	`ALOEInstallEventHandler`
	`ALOEGetEventHandler`
	`ALOERemoveEventHandler`
	`ALOEInstallPreDispatchHandler`
	`ALOEGetPreDispatchHandler`
	`ALOERemovePreDispatchHandler`
	`ALOEInstallObjectAccessor`
	`ALOEGetObjectAccessor`
	`ALOERemoveObjectAccessor`
	`ALOESetObjectCallbacks`
	`ALOEInstallCoercionHandler`
	`ALOEGetCoercionHandler`
	`ALOERemoveCoercionHandler`
	`ALOEGetCoercionHandler`
	`ALOEResolve`
	`ALOEDisposeToken`

Description of API Routines

ALOE Initialization and Termination

```
OSErr StartALOE();
```

Basic operation

This function initializes ALOE for the current process. It should do whatever OpenDoc initialization is necessary, including creating the session.

Inputs

Outputs

OSErr	result code

Errors Returned

noErr	Initialization Successful.
kALOEOpenDocNotPresent	OpenDoc Not present.

Pre conditions

none

Post conditions

ALOE should be ready to either load or create adaptor managers.

```
OSErr TerminateALOE();
```

Basic operation

This function Terminates ALOE. OpenDoc is shutdown, the session deleted. Calls to ALOE after TerminateALOE is called will result in default behavior or errors, depending on the call.

Inputs

none

Outputs

none

Errors Returned

noErr	Successful termination.
kALOEFailedToTerminate	Unable to shut down ALOE

Pre conditions

ALOE has been previously initialized. Any adaptor managers created since ALOE was started have been disposed.

Post conditions

ALOE is shut down.

Adaptor Managers

```
OSErr ALOENewAdaptorMgr(ALOEAdaptorMgr* newAdaptorMgr);
```

Basic operation

This routine creates a new, empty, adaptor manager. An adaptor manager corresponds to a document. This should be called for each document that can embed OpenDoc parts.

Inputs

none

Outputs

`newAdaptorMgr`	The new adaptor manager.

Errors returned

memFullErr Not enough free memory to create the adaptor manager.

Pre conditions

ALOE has been initialized with `StartALOE`.

Post conditions

The manager returned is ready to create and manage adaptors.

See also

`ALOENewAdaptorMgrFromFile, ALOENewAdaptorMgrFromMemory`

```
OSErr ALOEDisposeAdaptorMgr(ALOEAdaptorMgr adaptorMgr);
```

Basic operation

This routine shuts down the in memory presence of an adaptor manager.

Inputs

adaptorMgr The manager to dispose

Outputs

none

Errors Returned

noErr Success.

kALOEFailedToCloseManager Failure.

kALOEInvalidMgr Adaptor manager is not valid.

Pre conditions

Any adaptors created by the manager have been disposed.

Post conditions

The manager is no longer a valid argument to ALOE routines.

```
OSErr ALOEIsAdaptorMgrDirty(ODAdaptorMgr adaptorMgr, Boolean* changed,
                            Boolean* isSameSize);
```

Basic operation

This function checks the changed status of the given adaptor manager.

Inputs

adaptorMgr The manager to check

Outputs

changed Returns true if adaptor manager has changed

isSameSize Returns true if the size of storage used by adaptor manager has not changed since the last save.

Errors Returned

noErr Success.

kALOEInvalidMgr Adaptor manager is not valid.

Pre conditions

none

Post conditions

none

```
OSErr ALOENewHandleFromAdaptorMgr(ALOEAdaptorMgr adaptorMgr,
                      Handle* destination, Boolean allocateInTempMem);
```

Basic operation

This function allocates a handle and copies the manager and all adaptors to the handle. Only adaptors with corresponding persistent ids are copied; other adaptors

that have been created but not passed to `ALOEGetPersistentID` are not copied to the handle. Also, an adaptor is not copied to the handle if it is in the removed state, or was disposed while in the removed state. The caller is responsible for disposing of the handle.

Inputs

adaptorMgr	The manager to be stored into the handle
allocateInTempMem	`true` if the handle should be allocated in temp mem

Outputs

destination	The handle returned by this function

Errors Returned

noErr	Success.
memFullErr	Unable to allocate handle.
kALOEInvalidMgr	Adaptor manager is not valid.

Pre conditions

none

Post conditions

none

See also

`ALOEGetPersistentID, ALOESetAdaptorRemovedState`

```
OSErr ALOEWriteAdaptorMgrToFile(ALOEAdaptorMgr adaptorMgr, long refNum,
                                long* bytesWritten);
```

Basic operation

This function streams an adaptor manager and its contents to a file. Only adaptors with corresponding persistent ids are written; other adaptors that have been created but not passed to `ALOEGetPersistentID` are not written to the file. Also, an adaptor is not written to the file if it is in the removed state, or was disposed while in the removed state.

Inputs

adaptorMgr	The adaptor manager to write
refNum	The reference number of the file to write to

Outputs

bytesWritten	The number of bytes actually written to the file

Errors Returned

kALOEInvalidMgr	Adaptor manager is not valid.
wPrErr	Volume is write protected.
fLckdErr	File is locked.
eofErr	End of file.
fnOpnErr	File not open.

Pre conditions

File must be open and the file mark must be set.

Post conditions

none

See also

`ALOEGetPersistentID, ALOESetAdaptorRemovedState`

```
OSErr ALOENewAdaptorMgrFromFile(long refNum,
                                ALOEAdaptorMgr* newAdaptorMgr);
```

Basic operation

This function creates an adaptor manager, initializes it from disk, and returns it.

Inputs

refNum	The working directory number of the file to read the adaptor manager from

Outputs

newAdaptorMgr	The newly created adaptor manager

Errors Returned

eofErr	End of file.
fnOpnErr	File not open.
memFullErr	Unable to allocate memory.

Pre conditions

ALOE has been started, the file is open, and the file mark is set correctly.

Post conditions

The adaptor manager is initialized. Adaptors saved in the file can be retrieved from the manager by their persistent ids.

See also

`ALOEWriteAdaptorMgrToFile, ALOEGetAdaptor`

```
OSErr ALOENewAdaptorMgrFromMemory(Handle source,
                                  ALOEAdaptorMgr* newAdaptorMgr);
```

Basic operation

This function creates an adaptor manager, initializes it from memory, and returns it. The contents of the handle are copied to temp storage so that the caller can dispose of the handle or use it for other purposes after this call returns.

Inputs

source	The handle containing the manager data

Outputs

newAdaptorMgr	The created manager

Errors Returned

memFullErr	Unable to create manager.

Pre conditions

ALOE has been started.

Post conditions

The given handle can be immediately disposed since the call copies the contents to temp storage.

The adaptor manager is initialized. Adaptors saved in the file can be retrieved from the manager by their persistent ids.

See also

`ALOENewHandleFromAdaptorMgr, ALOEGetAdaptor`

```
OSErr ALOEGetPersistentID(ALOEAdaptor adaptor, ALOEPersistentID id);
```

Basic operation

Given an adaptor, return its persistent id. The persistent id can be used to retrieve the adaptor when the adaptor manager is reinstantiated from file or memory.

Inputs

adaptor	The adaptor.

Outputs

id	The returned persistent identifier

Errors Returned

kALOEInvalidAdaptor	This is not a valid adaptor.

Pre conditions

none

Post conditions

none

See also

`ALOEGetAdaptor`

```
OSErr ALOEGetAdaptor(ALOEAdaptorMgr adaptorMgr, ALOEPersistentID id,
                     WindowPtr window, ALOEAdaptor* newAdaptor);
```

Basic operation

This function takes a persistent id and returns its corresponding runtime adaptor, loading the adaptor from storage, if necessary. If an adaptor already exists for the argument persistent id, no new adaptor is created; the existing adaptor is returned. The application must eventually dispose of each adaptor by calling `ALOEDisposeAdaptor`.

Inputs

adaptorMgr	The manager in which the adaptor exists.
id	The adaptor's persistent id.
window	The window in which the adaptor will be embedded.

Outputs

newAdaptor	The returned adaptor.

Errors Returned

memFullErr	Unable to bring adaptor into memory.
kALOEInvalidPersistentID	The persistent id doesn't exist in the given adaptor manager.
kALOEInvalidMgr	Adaptor manager is not valid.

Pre conditions

none

Post conditions

The returned adaptor is internalized and is a valid argument to ALOE routines.

See also

`ALOEGetPersistentID`

```
ALOEAdaptorMgr ALOEGetAdaptorMgrFromAdaptor(ALOEAdaptor adaptor);
```

Basic operation

This function returns the manager which created the argument adaptor.

Inputs

adaptor	The adaptor.

Outputs

ALOEAdaptorMgr	The adaptor manager for the given adaptor.

Errors Returned

kALOEInvalidAdaptor	This is not a valid adaptor.

Pre conditions

none

Post conditions

none

Window Management

The functions `IsALOEWindow`, `IsALOEFloatingWindow`, and `ALOESizeWindow` can be called by any type of application. The functions `ALOEDragWindow` and `ALOEWindowActionDispatch` should be called by applications that don't support floating windows. The functions `ALOESetWindowCallback` and `ALOEDeactivateWindow` should be called by applications that have their own floating window support. If an application supports floating windows, then it must also support ALOE's floating windows.

```
Boolean IsALOEWindow(WindowPtr window);
```

Basic operation

This routine returns true if the given window was created by ALOE. Otherwise, it returns false.

Inputs

window	The WindowPtr of the window in question.

Outputs

Boolean	True if the window was created by ALOE (i.e., whether the window is created by an OpenDoc part editor).

Errors Returned

none

Pre conditions

none

Post conditions

none

```
Boolean IsALOEFloatingWindow(WindowPtr window);
```

Basic operation

This function returns true if the given window is a floating window created by ALOE. Otherwise, it returns false.

Inputs

window	The WindowPtr of the window in question.

Outputs

Boolean	True if the window is a floating window created by ALOE (i.e., created by an OpenDoc part editor).

Errors Returned

none

Pre conditions

none

Post conditions

none

```
OSErr ALOESizeWindow(WindowPtr window, short w, short h, Boolean fUpdate);
```

Basic operation

This function is used to resize a window that contains an adaptor. It lets ALOE and the embedded adaptors know about changes to the window size. If ALOE is not loaded the functionality is the same as `SizeWindow`.

Inputs

window	A pointer to the window record of the window to be resized.

w	The new window width, in pixels.
h	The new window height, in pixels.
fUpdate	A Boolean value that specifies whether any newly created area of the content region is to be accumulated into the update region (TRUE) or not (FALSE). You ordinarily pass a value of TRUE to ensure that the area is updated. If you pass FALSE, you're responsible for maintaining the update region yourself. For more information on adding rectangles to and removing rectangles from the update region, see the description of InvalRect on page 4-107 of *Inside Macintosh: Macintosh Toolbox Essentials* and ValidRect on page 4-108 of *Inside Macintosh: Macintosh Toolbox Essentials*.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

The window will be resized.

```
OSErr ALOEDragWindow(WindowPtr window, Point startPt,
        const Rect* boundsRect);
```

Basic operation

This function is used by container applications that don't support floating windows to drag a window. It is used in place of DragWindow. If the window is not a floating window it will be placed above all document windows but behind the floating windows.

Inputs

window	A pointer to the window record of the window to be dragged.
startPt	The location, in global coordinates, of the cursor at the time the user pressed the mouse button. Your application retrieves this point from the where field of the event record.
boundsRect	A rectangle, in global coordinates, that limits the region to which a window can be dragged. If the mouse button is released when the cursor is outside the limits of boundsRect, DragWindow returns without moving the window (or, if it was inactive, without making it the active window).

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

The window may be moved to a new position and the windows ordering in the window

list may change.

```
OSErr ALOEWindowActionDispatch(WindowPtr window, ALOEWindowAction action,
        Boolean param);
```

Basic operation

This function is used by container applications that don't support floating windows to cause window actions. `ALOEWindowActionDispatch` should be used instead of `ShowWindow`, `HideWindow` or `SelectWindow`. There are macros in ALOE.h that should be used in place of these Macintosh toolbox calls.

Inputs

window	A pointer to the window record of the window this action corresponds to.
action	The type of action for this window. The values may be kALOEAddWindow, kALOERemoveWindow, kALOEShowWindow, kALOEHideWindow, or kALOESelectWindow.
param	This value is currently unused and should always be set to false. It may be used in the future.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

ALOE will perform an action corresponding to the action value. If the value is kALOEShowWindow or kALOEHideWindow the window will be shown or hidden respectively. If the value is kALOESelectWindow the window will be selected.

```
OSErr  ALOESetWindowCallback(ALOEWindowCallbackUPP windowCallbackProc,
        void * contextPtr);
```

Basic operation

This function is used to set a call back to be called by ALOE every time a window action occurs. A window action is an event that may cause a change in the window list. The container application will install this callback if it is handling floating windows. ALOE will subsequently call this function when it needs a window action performed.

Inputs

windowCallbackProc	The pointer to the call back function.
context	This is a reference constant that will be passed to the call back each time.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

Callback function

```
void WindowActionCBProc(WindowPtr window, ALOEWindowAction
```



```
action, Boolean param, void* contextPtr);
```

The action parameter will have one of the following values: `kALOEAddWindow`, `kALOERemoveWindow`, `kALOEShowWindow`, `kALOEHideWindow`, or `kALOESelectWindow`. `kALOEAddWindow` tells the container application that ALOE has created a window. The param value will be set to true if the window is a floating window. The window will not necessarily be shown. `kALOERemoveWindow` tells the container application that ALOE has removed a previously added window. `kALOEShowWindow` tells the container application that ALOE wishes to have a window shown. `kALOEHideWindow` tells the container that ALOE wishes to hide a window. `kALOESelectWindow` tells the container application to select a window. The container application must subsequently perform the action ALOE requests.

```
OSErr ALOEDeactivateWindow(WindowPtr window);
```

Basic operation

This function is used by container applications that support floating windows to notify ALOE that a window has been deactivated.

Inputs

window	A pointer to the window record of the window to be deactivated.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

The window will be deactivated.

Events / Activation

```
OSErr ALOESetActivateApplicationCallback(ALOEActivateApplicationUPP proc,
        void *contextPtr);
```

Basic operation

This function is used to set a call back to be called by ALOE when the application needs to become active or inactive.

Inputs

proc	The pointer to the call back function.
context	This is a reference constant that will be passed to the call back each time.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

The call back function will subsequently be called by ALOE when ALOE needs the window.

Callback


```
void ActivateAppProc(Boolean activate, void* contextPtr);
```

See also

`ALOEApplicationActive`

```
OSErr ALOEApplicationActive();
```

Basic operation

This function deactivates any active adaptor or embedded part. The activate application callback, if installed by the application, is NOT called.

Inputs

none

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOESetActivateApplicationCallback`

```
OSErr ALOEActivateAdaptor(ALOEAdaptorView adaptor);
```

Basic operation

This function is used to get ALOE to activate the given adaptor. In general, the active state of the adaptor is handled by ALOE depending on user action (e.g., mouse click). When the application wants to directly activate an adaptor, this call should be used.

Inputs

adaptor	The adaptor to activate.

Outputs

none

Errors Returned

kALOEInvalidAdaptor	This is not a valid adaptor.

Pre conditions

none

Post conditions

none

```
OSErr ALOESetDoesPropagateEvents(ALOEAdaptor adaptor, Boolean
        doesPropagate);
```

Basic operation

This function is used to get set the DoesPropagate flag of an adaptor. If the adaptor's DoesPropagate flag is set to true then the adaptor will propagate events that it does not handle up the embedded part hierarchy. If the flag is set to true, then the embedded adaptor will swallow unhandled events.

Inputs

adaptor	The adaptor whose DoesPropagate flag will be changed.
doesPropagate	The value to which to change the adaptor's

DoesPropagate flag.

Outputs

none

Errors Returned

kALOEInvalidAdaptor	This is not a valid adaptor.

Pre conditions

none

Post conditions

none

```
OSErr ALOESetBaseMenuBar(short mbarResource);
```

Basic operation

Use this function to set ALOE's base menu bar from a resource id.

Inputs

mbarResource	The resource id of the mbar resource to use as the base.

Outputs

none

Errors Returned

memFullErr	Not enough memory to load the resource
resNotFound	Could not load the resource

Pre conditions

none

Post conditions

All adaptors created after this call is made will copy their menu bar from the new base menu bar.

See also

`ALOESetBaseMenuBar`

OpenDoc Programmer's Guide, Chapter 6, "Setting up Menus".

```
OSErr ALOESetBaseMenuBarHandle(Handle mbarHandle);
```

Basic operation

Use this function to set ALOE's base menu bar from a handle.

Inputs

mbarHandle	The handle to the mbar resource to use as the base.

Outputs

none

Errors Returned

kInvalidMBarHandle	The handle is not a valid menu bar.

Pre conditions

none

Post conditions

All adaptors created after this call is made will copy their menu bar from the new base menu bar.

See also

`ALOESetBaseMenuBarWithResource`

OpenDoc Programmer's Guide, Chapter 6, "Setting up Menus".


```
OSErr ALOERegisterMenuCommand(short menuID, short menuItem, long
        commandID);
```

Basic operation

This function is used to register menu commands with ALOE. This should be done minimally for the standard commands: Print, Page Setup, Cut, Copy, Paste, Clear. ALOE constants should be used for the standard commands. Additionally, if the application would like to use ALOE's menu handling facilities, any other application commands can be registered with this call. Applications can use ALOEGetMenuCommand to get a command for a menu/item pair.

Inputs

menuID	The menu for the command be registered.
menuItem	The item in the menu for the command being registered.
commandID	The command ID to be registered for the given menu/item pair.

Outputs

none

Errors Returned

kInvalidMenuItemPair	The menu and item don't exist

Pre conditions

none

Post conditions

none

See also

`ALOEGetMenuCommand`

```
OSErr ALOEGetMenuCommand(short menuID, short menuItem, long* commandID);
```

Basic operation

This function returns a menu command given a menu/item pair. The command ID must have been previously registered by a call to `ALOERegisterMenuCommand`, or have been registered by ALOE.

Inputs

menuID	The menu containing the desired command.
menuItem	The item containing the desired command.
commandID	The command ID returned for the given menu/item pair.

Outputs

none

Errors Returned

kUnregisteredCommand	No command has been registered for the given menu/item pair.
kInvalidMenuItemPair	The menu and item don't exist

Pre conditions

none

Post conditions

none

See also

`ALOERegisterMenuCommand`

```
OSErr ALOEAdjustMenus();
```


Basic operation

This function is called to allow embedded adaptors (mainly the active adaptor) to set up their menus before the application calls MenuSelect.

Inputs

none

Outputs

none

Errors Returned

none

Pre conditions

This call should be made after an event is received by the application and is determined to be in the menu bar. The call should be made before MenuSelect is called.

Post conditions

none

```
Boolean ALOEIsUndoClear();
```

Basic operation

This function is used by the container application to determine if any ALOE embedded adaptors have any pending undo/redo actions. This call should be used to determine whether or not the container application should munge and/or enable the "undo" item in the edit menu. In the "false" case the edit menu text should be set up by ALOE when ALOEAdjustMenus is called.

Inputs

none

Outputs

Boolean	False if ALOE has pending undo/redo actions. True if ALOE does not have any pending undo/redo.

Errors Returned

none

Pre conditions

none

Post conditions

none

```
OSErr ALOEClearUndo();
```

Basic operation

Forces ALOE to clear its undo stack.
This is usually called when the user does an action which cannot be undone in the application.

Inputs

none

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

The Undo stack maintained by ALOE for the embedded parts is empty.


```
Boolean ALOEHandleEvent(EventRecord* event);
```

Basic operation

This function is called to allow ALOE a chance to handle an event. It should be called before the application processes the event. However, the event will be modified when ALOEHandleEvent returns. The application should save the event record off and pass a copy to ALOEHandleEvent, using the original for it's own event processing.

Inputs

event	The pointer to the event record returned from WaitNextEvent.

Outputs

Boolean	true if the event was handled by ALOE.

Errors Returned

none

Pre conditions

none

Post conditions

none

```
RgnHandle ALOEGetSleepRegion(RgnHandle appSleepRegion);
```

Basic operation

This function gives ALOE a chance to "fix-up" the application's sleep region before passing to WaitNextEvent.

Inputs

appSleepRegion	sleep region calculated by the container applications.

Outputs

RgnHandle	sleep region adjusted by ALOE

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOEGetEventMask, ALOEGetSleepTime`

```
EventMask ALOEGetEventMask(EventMask appMask);
```

Basic operation

This function gives ALOE a chance to "fix-up" the application's event mask before passing to WaitNextEvent.

Inputs

appMask	The container application's event mask.

Outputs

EventMask	event mask adjusted by ALOE.

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOEGetSleepRegion, ALOEGetSleepTime`

```
UInt32 ALOEGetSleepTime(UInt32 appSleepTime);
```

Basic operation

This function gives ALOE a chance to “fix-up” the application’s sleep time before passing to WaitNextEvent.
This will allow embedded adaptors to get idle time correctly.

Inputs

appSleepTime	The container application’s sleep time.

Outputs

UInt32	sleep time adjusted by ALOE.

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOEGetEventMask, ALOEGetSleepRegion`

```
Boolean ALOEIdle(void);
```

Basic operation

This function allows embedded adaptors to get idle time with the correct *parameters, without having to create a "dummy" idle event.*

Inputs

none

Outputs

Boolean	true if the event was handled by ALOE.

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOEHandleEvent`

Adaptors

```
OSErr ALOENewAdaptor(ALOEAdaptorMgr adaptorMgr, WindowPtr window,
        ALOEISOString kind, Ptr dataPtr, long dataSize, RgnHandle shape,
        ALOEAdaptor* newAdaptor);
```

Basic operation

This routine creates a new adaptor of a given kind. Optionally, the adaptor can be created with initial content. The application must eventually dispose of the returned adaptor by calling `ALOEDisposeAdaptor`.

Inputs

adaptorMgr	The adaptor manager to which the new adaptor will be added.
window	The window to contain the new adaptor.
kind	The kind of data the adaptor will edit.
dataPtr	(Optional) A pointer to the data to stuff into the new adaptor.
dataSize	The size in bytes of dataPtr.
shape	The shape of the new adaptor

Outputs

newAdaptor	The newly created adaptor.

Errors Returned

kALOEInvalidAdaptorMgr	Adaptor manager is not valid.
memFullErr	Not enough memory available to create adaptor view.

Pre conditions

none

Post conditions

The new adaptor is a valid argument to ALOE routines.

See also

`ALOEGetAdaptor, ALOENewSynchronizedAdaptor, ALOEDuplicateAdaptor, ALOEGetKindFromResType`

```
OSErr ALOENewSynchronizedAdaptor(ALOEAdaptor fromAdaptor,
        WindowPtr window, ALOEAdaptor* newAdaptor);
```

Basic operation

This routine is used to create a new adaptor from an existing adaptor. The new adaptor will display the same underlying content as the original adaptor; that is, changes the user makes to `fromAdaptor` will be reflected in views showing the new adaptor, and vice versa. A persistent id can be created for the new adaptor that is distinct from one created for the original adaptor. The window argument must be a different than the original adaptor'swindow. The new adaptor has the same shape as the original adaptor.

The application must eventually dispose of the returned adaptor by calling `ALOEDisposeAdaptor`.

Inputs

fromAdaptor	The adaptor from which to generate the new adaptor.
window	The window in which the adaptor will be displayed.

Outputs

newAdaptor	The adaptor created as a result of the new.

Errors Returned

kALOEInvalidAdaptor	This is not a valid adaptor.
memFullErr	Not enough memory available to create adaptor.

Pre conditions

none

Post conditions

The new adaptor is a valid argument to ALOE routines.

See also


```
ALOEGetAdaptor, ALOENewAdaptor, ALOEDuplicateAdaptor
```

```
OSErr ALOEBeginDuplicateClone(ALOEAdaptorMgr fromAdaptorMgr,
        ALOEAdaptorMgr toAdaptorMgr, WindowPtr window);
```

Basic operation

This routine begins a transaction for duplicating adaptors. Until this transaction is completed by calling ALOEEndClone or ALOEAbortClone, the application may make calls to ALOEDuplicateAdaptor to duplicated adaptors in the same adaptor manager, or from one adaptor manager to another.

When adaptors are duplicated into a different adaptor manager, the scripting ids of the new adaptors will be different than their originals.

Inputs

`fromAdaptorMgr`	The adaptor manager containing adaptors to be duplicated.
`toAdaptorMgr`	The adaptor manager to contain the new adaptors.
window	The window in which the new adaptors will be displayed.

Outputs

none

Errors Returned

noErr	No error
kALOEInvalidAdaptor	This is not a valid adaptor.
memFullErr	Not enough memory available to create adaptor.

Pre conditions

A clone transaction is not in progress.

Post conditions

If the return value is noErr, a duplicate clone transaction has begun.

See also

```
ALOEDuplicateAdaptor, ALOEEndClone, ALOEAbortClone
```

```
OSErr ALOEDuplicateAdaptor(ALOEAdaptor fromAaptor, ALOEAdaptor*
        newAdaptor);
```

Basic operation

This function creates a new adaptor from an existing adaptor. It must be called during a duplicate transaction begun by calling ALOEBeginDuplicateClone. The new adaptor is created by the adaptor manager and in the window specified in the preceeding call to ALOEBeginDuplicateClone. Unlike `ALOENewSynchronizedAdaptor`, this routine returns an adaptor which is initialized to a copy of the existing adaptor's content, but is completely independent of the original. Changes to the original adaptor do not affect the new adaptor. The new adaptor has the same shape as the original adaptor.

If this funtion returns noErr, newAdaptor will be set to a valid adaptor. However, this adaptor should not be passed to other ALOE routines until the duplicate transaction has been completed by calling ALOEEndClone or ALOEAbortClone. Doing so will result in the error kALOEErrAdaptorNotInitialized. This includes ALOENewAdaptorView, ALOEGetPersistentID, and even ALOEDisposeAdaptor, for example. If the clone transaction is sucessfully completed by calling EndClone, the adaptor may be used as an argument to any ALOE routine. However, if the clone transaction is aborted by calling ALOEAbortClone, (or if ALOEEndClone returns an

error), the adaptor can only be passed to ALOEDisposeAdaptor. Other routines will return the error kALOEInvalidAdaptor.

If the newAdaptor value returned is non-null, the application must eventually dispose of the returned adaptor (after the duplicate transaction has been completed) by calling `ALOEDisposeAdaptor`.

Inputs

fromAdaptor	The adaptor from which to generate the new adaptor.

Outputs

newAdaptor	The adaptor created as a result of the duplicate.

Errors Returned

noErr	No error
kALOEInvalidAdaptor	Argument is not a valid adaptor.
memFullErr	Not enough memory available to create adaptor.

Pre conditions

A duplicate clone transaction initiated by ALOEBeginDuplicateClone is still in progress.

Post conditions

If the return value is noErr, a new adaptor is returned.

See also

`ALOEBeginDuplicateClone, ALOEEndClone, ALOEAbortClone, ALOEGetAdaptor, ALOENewAdaptor, ALOENewSynchronizedAdaptor`

```
OSErr ALOEDisposeAdaptor(ALOEAdaptor adaptor);
```

Basic operation

This routine frees the in-memory structures for the given adaptor. If the adaptor is in the removed state, any persitent id created for it is no longer valid, and the adaptor will not be copied in a subsequent call to `ALOEWriteAdaptorMgrToFile` or `ALOENewHandleFromAdaptorMgr`.

Note that multiple calls to `ALOEGetAdaptor` with the same persistent id will return the same adaptor. It is invalid to dispose of an adaptor twice.

Inputs

adaptor	The adaptor to dispose.

Outputs

none

Errors Returned

kALOEInvalidAdaptor	Argument is not a valid adaptor.
kALOEErrAdaptorNotInitialized	Argument adaptor was created by a clone transaction that is still in progress.

Pre conditions

none

Post conditions

If the return value is noErr, the argument adaptor is no longer valid.

See also

`ALOESetAdaptorRemovedState, ALOEWriteAdaptorMgrToFile, ALOENewHandleFromAdaptorMgr`

```
OSErr ALOESetAdaptorRemovedState(ALOEAdaptor adaptor, Boolean removed);
```


Basic operation

This routine marks the adaptor as being removed from (or restored to) the content maintained by its adaptor manager. This routine should be called with the argument `true` when the application cuts or deletes the adaptor, after disposing any adaptor views displaying the adaptor. If the application later undoes the deletion (and has not disposed of the adaptor), it can call this routine with the argument `false` to restore the adaptor, and then recreate adaptor views to redisplay it.

If the adaptor is disposed when in the removed state, any persitent id created for it is no longer valid and must not be used by the application. In addition, if an adaptor is in the removed state, or has been disposed while in the removed state, it will NOT be copied by `ALOEWriteAdaptorMgrToFile` or `ALOENewHandleFromAdaptorMgr`.

Inputs

adaptor	The adaptor.
removed	True, to put the adaptor in the removed state, or False, to restore the adaptor to its normal state.

Outputs

none

Errors Returned

kALOEInvalidAdaptor	This is not a valid adaptor.

Pre conditions

Any adaptor views displaying the adaptor have been disposed.

Post conditions

none

See also

`ALOEDisposeAdaptor, ALOEWriteAdaptorMgrToFile, ALOENewHandleFromAdaptorMgr`

```
OSErr ALOESetAdaptorShapeRequestCallback(
                ALOEAdaptorShapeRequestUPP proc,
                void *contextPtr);
```

Basic operation

This function is used to set the AdaptorShapeRequest call back. This call back is invoked when an adaptor requests frame shape negotiation.

If no callback is installed, requests to change shape are denied.

Inputs

proc	The call back function pointer.
contextPtr	A reference constant passed back to each invocation of the call back.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

Callback

```
                void AdaptorShapeRequest(ALOEAdaptor adaptor,
                                         RgnHandle theRequestedRgn,
                                         oid* contextPtr);
```



```
OSErr ALOEGetAdaptorShape(ALOEAdaptor adaptor, RgnHandle adaptorShape);
```

Basic operation

This routine returns the adaptor's shape in the supplied region handle, which the application should allocate by calling the toolbox routine `NewRgn`().

Inputs

adaptor	The adaptor.

Outputs

adaptorShape	The current shape of the adaptor

Errors Returned

kALOEInvalidAdaptor	This is not a valid adaptor.

Pre conditions

none

Post conditions

none

See also

`ALOESetAdaptorShape`

```
OSErr ALOESetAdaptorShape(ALOEAdaptor adaptor, RgnHandle adaptorShape);
```

Basic operation

This routine sets the shape of the adaptor to that of the argument region. The origin of the adaptorShape is irrelevant, however, the shape actually assigned will be normalized such that the top left of the region's bounding box is at (0,0). The application should dispose of the argument region handle.

Inputs

adaptor	The adaptor.
adaptorShape	The new shape of the adaptor.

Outputs

none

Errors Returned

kALOEInvalidAdaptor	This is not a valid adaptor.

Pre conditions

none

Post conditions

The adaptor's shape has changed.

See also

`ALOEGetAdaptorShape`

```
OSErr ALOEShowAdaptorInfo(ALOEAdaptorView view);
```

Basic operation

This function displays the OpenDoc Part Info dialog. Any changes to dialog settings made by the user are performed by this routine.

Inputs

view	The adaptor view.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

```
ALOEISOString ALOEGetKindFromResType(ResType type);
```

Basic operation

This function returns the ISO type corresponding to the specified scrap type. This routine can be used to convert a four-character scrap type to an OpenDoc kind to create a new embedded adaptor by calling `ALOENewAdaptor`. The application should dispose of the return value by calling the toolbox routine `DisposePtr`.

Inputs

type	The scrap type.

Outputs

result	The ISO type corresponding to `type`

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOENewAdaptor`

Adaptor Views

```
OSErr ALOENewAdaptorView(ALOEAdaptor adaptor, GrafPtr port, Point offset,
        THPrint printJob, ALOEAdaptorView* newView);
```

Basic operation

This routine creates a new adaptor view on a given adaptor. Multiple views may be created on an adaptor to support application features such as split views or spanning an adaptor across page boundaries.

If the adaptor was created by calling ALOECloneAdaptorToContent, this routine must not be called until the clone transaction started by ALOEBeginScrapCloneor ALOEBeginDragCloneis successfully completed by calling ALOEEndClone.

Inputs

adaptor	The adaptor to which the new view will be added.
port	The port to contain the new view; NULL to use the adaptor's window. If the view is a printing view the value should be a printing port. If this value is not NULL and printJob is NULL the view is assumed to be offscreen.
offset	The view's offset from the port's origin.
printJob	The print job (if any) associated with this view.

Outputs

newView	The newly created adaptor view.

Errors Returned

kALOEInvalidAdaptor	Argument is not a valid adaptor.
kALOEErrAdaptorNotInitialized	Argument adaptor was created by a clone

transaction that is still in progress.

memFullErr — Not enough memory available to create adaptor view.

Pre conditions

none

Post conditions

none

See also

`ALOECloneAdaptorToContent`

```
OSErr ALOEDisposeAdaptorView(ALOEAdaptorView view);
```

Basic operation

This routine frees the in-memory structures for the given adaptor.

Inputs

view — The adaptor view to dispose.

Outputs

none

Errors Returned

kALOEInvalidAdaptorView — This is not a valid adaptor view.

Pre conditions

The view was created by `ALOENewAdaptorView`.

Post conditions

The view is no longer a valid argument to ALOE routines.

See also

`ALOENewAdaptorView`

```
OSErr ALOESetAdaptorViewOffset(ALOEAdaptorView view, Point offset);
```

Basic operation

This function sets the horizontal and vertical offset of the given view. The offset should be given in local coordinates of the view's port.

Inputs

view — The view to change the size of.

offset — New offset from the port's origin.

Outputs

none

Errors Returned

kALOEInvalidAdaptorView — The given adaptor view is invalid.

Pre conditions

none

Post conditions

none

```
OSErr ALOEGetAdaptorViewOffset(ALOEAdaptorView view, Point* offset);
```

Basic operation

This function returns the horizontal and vertical offset of the given view in local coordinates of the view's port.

Inputs

view — The view.

Outputs

offset	The view's offset.

Errors Returned

kALOEInvalidAdaptorView	The given adaptor view is invalid.

Pre conditions

none

Post conditions

none

```
OSErr ALOESetAdaptorViewScaleFactor(ALOEAdaptorView view,
                                    Fixed horzScaleFactor,
                                    Fixed vertScaleFactor);
```

Basic operation

This function changes the scaling factor of an adaptor view. The horizontal and vertical scaling factors are specified separately. The scaling factors are specified as 32-bit fixed-point values; 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.

Any scaling information is not persistent and is lost when the view is destroyed. The view is scaled in place, so the view's offset is not affected by the scaling. Likewise, the size of the view is not affected by scaling.

Inputs

horzScaleFactor	The new horizontal scaling factor.
vertScaleFactor	The new vertical scaling factor.

Outputs

none

Errors Returned

kALOEInvalidAdaptorView	The given adaptor view is invalid.

Pre conditions

none

Post conditions

none

```
OSErr ALOEGetAdaptorViewScaleFactor(ALOEAdaptorView view,
                                    Fixed* horzScaleFactor,
                                    Fixed* vertScaleFactor);
```

Basic operation

This function returns the horizontal and vertical scale factor of the given adaptor view. If no scaling has been set for this view then zero is be returned.

Inputs

horzScaleFactor	The new horizontal scaling factor.
vertScaleFactor	The new vertical scaling factor.

Outputs

none

Errors Returned

kALOEInvalidAdaptorView	The given adaptor view is invalid.

Pre conditions

none

Post conditions

none

```
OSErr ALOEDrawAdaptorView(ALOEAdaptorView view, RgnHandle viewClip);
```

Basic operation

Cause the view given to be drawn. If viewClip is a null region handle, the last clipping region specified is used. If no clipping region has been specified in a previous call to this routine for the same adaptor view, the clip shape defaults to the adaptor shape repositioned by the view's current offset.

Inputs

view	The view to draw.
viewClip	The region to use to clip out the view in local coordinates of the view's port. May be NULL.

Outputs

none

Errors Returned

kALOEInvalidAdaptorView	The given adaptor view is invalid.

Pre conditions

none

Post conditions

none

```
ALOEAdaptor ALOEGetAdaptorFromAdaptorView(ALOEAdaptorView view);
```

Basic operation

This function returns the adaptor which created the view.

Inputs

view	The adaptor view for which we are seeking an adaptor.

Outputs

ALOEAdaptor	The adaptor for the given view.

Errors Returned

kALOEInvalidAdaptorView	This is not a valid adaptor view.

Pre conditions

none

Post conditions

none

User Interface

```
OSErr ALOESetAdjustActiveBorderShapeCallback(
                ALOEAdjustActiveBorderShapeUPP proc, void *contextPtr);
```

Basic operation

This function is used to set the AdjustActiveBorder call back. This call back is invoked when an adaptor becomes active. When the adaptor becomes active, the Human Interface guidelines call for an active border being drawn around it. The drawing of this border is handled by ALOE. This callback gives the container application to adjust the active border shape so that any obscured area (probably due to application's content objects) can be removed.

If no callback is installed, the shape of the adaptor will be used.

Inputs

proc	The call back function pointer.
contextPtr	A reference constant passed back to each invocation of the call back.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

Callback

```
void AdjustActiveBorderProc(ALOEAdaptorView view, RgnHandle
      shape, void* contextPtr);
```

See also

OpenDoc Programmer's Guide, Chapter 4, "Managing the Active Frame Border".

```
Boolean ALOEGetAdaptorViewSelectState(ALOEAdaptorView view);
```

Basic operation

This function returns the selected state of an adaptor view.

Inputs

view	The adaptor view.

Outputs

result	A value of true indicates the view is selected.

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOESetAdaptorViewSelectState`

```
OSErr ALOESetAdaptorViewSelectState(ALOEAdaptorView view,
                                    Boolean selected);
```

Basic operation

This function is used to set the selected state of an adaptor view. This is called with true when the adaptor view falls in the selection of the application. When the adaptor view is no longer in the selection, the application should call this function with false. This is necessary because the adaptor view may be drawn differently if it is in a selection.

Inputs

view	The adaptor view to change.
selected	A value of true indicates selected.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

```
ALOEHighlightState ALOEGetAdaptorViewHighlightState(ALOEAdaptorView view);
```

Basic operation

This function returns the highlight state of an adaptor view.

Inputs

view	The adaptor view.

Outputs

result	The highlight state of the view.

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOESetAdaptorViewHighlightState`

```
OSErr ALOESetAdaptorViewHighlightState(ALOEAdaptorView view,
        ALOEHighlightState highlighted);
```

Basic operation

This function is used to set the highlight state of an adaptor view. There are three possible highlight states:

```
kALOENoHighlight
kALOEFullHighlight
kALOEDimHighlight
```

Inputs

view	The adaptor view to change.
state	The highlight state of the view.

Outputs

none

Errors Returned

kALOEInvalidAdaptorView	This is not a valid view.
kALOEInvalidState	This is not a valid highlight state.

Pre conditions

none

Post conditions

none

See also

These highlight states correspond to OpenDoc's highlight states. For more information, see OpenDoc Programmer's Guide Chapter 4, "Basic Drawing".

```
OSErr ALOEApplicationModal(Boolean requestModalFocus);
```


Basic operation

This call notifies ALOE that the application is either entering or leaving a modal state. For example, when the application is about to put up a modal dialog, it should call this function with true. Upon exiting the modal dialog, it should call this function again with false.

Inputs

requestModalFocus	True indicates the container is becoming modal.

Outputs

none

Errors Returned

kALOECannotRelinquishModalState	ALOE cannot relinquish the modal state.

Pre conditions

none

Post conditions

none

```
OSErr ALOESetAdaptorBundleState(ALOEAdaptor adaptor,
        Boolean bundled);
```

Basic operation

This function sets the bundle state of the given adaptor.

Inputs

adaptor	The adaptor to mark as bundled.
bundled	A value of true indicates the adaptor should be bundled.

Outputs

none

Errors Returned

kALOEInvalidAdaptor	The given adaptor is invalid.

Pre conditions

none

Post conditions

none

See also

OpenDoc Programmer's Guide, Chapter 13, "Bundling Frames".

```
Boolean ALOEGetAdaptorBundleState(ALOEAdaptor adaptor);
```

Basic operation

This function sets the bundle state of the given adaptor.

Inputs

adaptor	The adaptor to mark as bundled.

Outputs

Boolean	Returns true if the adaptor is bundled, false if not.

Errors Returned

kALOEInvalidAdaptor	The given adaptor is invalid.

Pre conditions

none

Post conditions

none

See also

OpenDoc Programmer's Guide, Chapter 13, "Bundling Frames".

```
OSErr ALOEGetAdaptorViewPicture(ALOEAdaptorView view,
                                PicHandle* image);
```

Basic operation

This function calls ALOE to get a static image of the given adaptor view, returned in the form of a picture handle.

Inputs

view	The adaptor view for which to get the image.

Outputs

image	The handle returned containing the image.

Errors Returned

kALOEInvalidAdaptorView	The given adaptor view is invalid.
kALOENoImageAvailable	A cached presentation cannot be generated.
memFullErr	Not enough memory to create the picture.

Pre conditions

none

Post conditions

none

```
OSErr ALOESetAdaptorImageUpdatedCallback(AdaptorImageUpdatedUPP proc,
                                         void *contextPtr);
```

Basic operation

This function allows the container application to set a call back allowing ALOE to notify the container when the adaptor's image has been changed.
This call back is only necessary when the container application is forcing the adaptor to draw off screen.

Inputs

proc	The call back function pointer.
contextPtr	The reference constant that will be passed back to the call back routine.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

Callback

```
void AdaptorImageUpProc(ALOEAdaptorView view, void*
     contextPtr);
```

```
OSErr ALOESetAdaptorChangedCallback(ALOEAdaptorChangedUPP proc, void
      *contextPtr);
```

Basic operation

This function sets up a call back for ALOE to notify the container application when an

embedded adaptor changes. This call back is used for compatibility with the Edition Manager. With this callback, the application can update the subscriber when an embedded adaptor changes.

Inputs

proc	A pointer to the call back function.
contextPtr	This reference constant will be passed each time the call back is made.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

Callback

```
void AdaptorChangedProc(ALOEAdaptorView view,
                        void* contextPtr);
```

```
OSErr ALOESetRevealAdaptorCallback(ALOERevealAdaptorUPP proc,
                                   void *contextPtr);
```

Basic operation

This function sets up a call back for ALOE to notify the container application when ALOE needs an embedded adaptor shown in an adaptor view. This call back is associated with the OpenDoc linking recipes. When a source link in an embedded adaptor (or part) needs to be revealed, this callback will be triggered.

Inputs

proc	A pointer to the call back function.
contextPtr	This reference constant will be passed each time the call back is made.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

Callback

```
void RevealAdaptorProc(ALOEAdaptor adaptor,
      void* contextPtr);
```

If the application has not created an adaptor view for the argument adaptor, it should do so and scroll its content, if necessary, to make the view visible.

See also

OpenDoc Programmer's Guide, Chapter 8, "Revealing the Source of a Link".

Clipboard - Scrap Manager cover routines

In order to make OpenDoc and the clipboard work more seamlessly together, ALOE provides a set of cover routines for existing toolbox calls. Clients of ALOE can simply replace their calls to the

toolbox with corresponding ALOE calls. If ALOE or OpenDoc is not present, these ALOE calls work exactly the same way as their counterpart in the toolbox.

```
PScrapStuff ALOEInfoScrap();
```

Basic operation

This call is an ALOE cover to the InfoScrap toolbox call.

Inputs

none

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOEGetScrapCount, ALOEScrapCountStillValid`

```
long ALOEZeroScrap();
```

Basic operation

This call is an ALOE cover to the ZeroScrap toolbox call.

Inputs

none

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

```
long ALOEPutScrap(long length, ResType theType, Ptr source);
```

Basic operation

This call is an ALOE cover to the `PutScrap` toolbox call.

The source parameter may be null to "promise" a scrap type. The length parameter is ignored in this case. The application must have already called `ALOESetWriteScrapCallback` to install a WriteScrap callback. The callback will only be invoked if the scrap data is needed before the scrap is changed by ALOE, the process is suspended, or the application calls `ALOEUpdateScrap`. Once a scrap type has been promised, the application may call `ALOEPutScrap` to replace the promise with actual data, if the scrap count has not changed since the promise was written. It is invalid to write scrap data twice.

Data may be promised before `StartALOE` is called.

Inputs

	length	The length of the source ptr.
	theType	The scrap format type of the data to be written to the scrap.
	source	A pointer to the data to put on the scrap.
Outputs		
	long	The function result.
Errors Returned		
	kALOEErrNoWriteScrapCallback	The WriteScrap callback has not been installed.
	kALOEErrScrapDataPresent	The application attempted to replace existing data on the scrap

Pre conditions

none

Post conditions

none

See also

`ALOESetWriteScrapCallback, ALOEGetScrapCount, ALOEScrapCountStillValid`

```
long ALOEGetScrap(Handle hDest, ResType theType, long *offset);
```

Basic operation

This call is an ALOE cover to the GetScrap toolbox call.

Inputs		
	hDest	The destination for the contents of the scrap.
	theType	The type of data requested.
Outputs		
	offset	Returns the location of the data as offset from beginning of the desk scrap.
	long	The function result.
Errors Returned		

none

Pre conditions

none

Post conditions

none

```
long ALOEUnloadScrap();
```

Basic operation

This call is an ALOE cover to the UnloadScrap toolbox call.

Inputs		
	none	
Outputs		
	long	The function result.
Errors Returned		

none

Pre conditions

none

Post conditions

none

```
long ALOELoadScrap();
```

Basic operation

This call is an ALOE cover to the LoadScrap toolbox call.

Inputs

none

Outputs

long — The function result.

Errors Returned

none

Pre conditions

none

Post conditions

none

Clipboard

These functions handle the copying of the adaptors between the application and the clipboard: `ALOEEmbedScrap`, `ALOEBeginScrapClone`, `ALOECloneAdaptorFromContent`, `ALOECloneAdaptorToContent`, `ALOEEndClone`, and `ALOEAbortClone`. The latter four are also used for Drag and Drop

```
short ALOEGetScrapCount();
```

Basic operation

An efficient means to get the current scrap count. This should be used in place of `ALOEInfoScrap` when only the scrap count field is desired.

Inputs

none

Outputs

result — `the current scrap count.`

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOEInfoScrap`

```
Boolean ALOEScrapCountStillValid(short scrapCount);
```

Basic operation

Returns true if the argument scrap count is the current scrap count. Applications can use this routine to decide if their private scrap is valid or stale, or if they need to preemptively fulfill a promise for scrap data.

Inputs

none

Outputs

result	`true` if the argument scrap count is still the current scrap count, and `false` otherwise.

Errors Returned

none

Pre conditions

none

Post conditions

none

See also

`ALOEInfoScrap, ALOEPutScrap`

```
OSErr ALOESetWriteScrapCallback(ALOEWriteScrapUPP writeDataProc, void
        *contextPtr);
```

Basic operation

An application is required to install a WriteScrap callback if it calls `ALOEPutScrap` with a null data pointer to promise data. A WriteScrap callback may be installed before calling `StartALOE`. Once installed, the WriteScrap callback can be changed by calling this routine again. When promised data is needed, the most recently installed WriteScrap callback is invoked.

The WriteScrap callback may be invoked from any ALOE routine. In particular, it may be invoked from `ALOEPutScrap` when the data pointer is null but the data is needed immediately.

Inputs

`writeDataProc`	A pointer to the call back function.
contextPtr	This reference constant will be passed each time the call back is made.

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

Callback

```
OSErr WriteScrapProc(ResType theType, void *contextPtr);
```

Basic operation

This application callback should call `ALOEPutScrap` with a non-null source pointer to write the actual data for the scrap type `theType`. The callback should not call `ALOEZeroScrap`. If an error is returned, and ALOEPutScrap has not been called to replace the data, the data length will be zero.

See also

`ALOEPutScrap`

```
OSErr ALOEUpdateScrap();
```

Basic operation

Forces synchronization of the platform scrap from the ALOE clipboard. Applications should call ALOEUpdateScrap before calling a routine that accesses the platform

scrap directly, such as TextEdit on the Macintosh. Only content that has a platform representation is copied to the scrap.

Inputs

none

Outputs

none

Errors Returned

none

Pre conditions

none

Post conditions

none

```
OSErr ALOEBeginScrapClone(ALOEAdaptorMgr adaptorMgr, WindowPtr window,
                          ALOECloneKind kind);
```

Basic operation

Used by the application to exchange data including embedded adaptors. This call is used to begin a transaction for cloning embedded adaptors to the clipboard.

There must be a matching `ALOEEndClone` or `ALOEAbortClone` (in case of error) for each call to `ALOEBeginScrapClone`.

Inputs

adaptorMgr	The adaptor manager containing adaptors to be copied to the clipboard, or to receive adaptors copied from the clipboard.
window	When kind is `kALOEPaste`, the window in which to embed adaptors; otherwise, this parameter is ignored.
kind	The type of clone (`kALOECut`, `kALOECopy`, or `kALOEPaste`).

Outputs

none

Errors Returned

kALOEErrCloneInProgress	A clone transaction is already in progress.
kALOEInvalidCloneKind	Invalid clone kind.
kALOEInvalidMgr	Adaptor manager is not valid.
kALOEErrInvalidNilWindow	The window argument must be non-nil.

Pre conditions

A clone transaction is not in progress.

Post conditions

A scrap clone transaction has begun.

See also

For interacting with Drag Manager, see `ALOEBeginDragClone`.

```
OSErr ALOECloneAdaptorFromContent(ALOEAdaptor adaptor, Point offset,
                          ALOEClonedID clonedId);
```

Basic operation

This routine is used to clone an adaptor to the ALOE clipboard or to a drag reference. A clone transaction must be in progress. The offset argument is used by ALOE to create an additional content kind, a frame list, to facilitate interchange with other ALOE applications and OpenDoc parts. The offset should specify the position of the cloned

adaptor relative other cloned adaptors, so that geometric relationships between the adaptors can be preserved by a destination. Typically, the offset is that of an adaptor view displaying the adaptor.

The returned id can be stored in data written by the application using `ALOEPutScrap` (for a scrap clone transaction) or `ALOEAddDragItemFlavor` (for a drag and drop clone transaction) and later used to recover the adaptor using `ALOECloneAdaptorToContent`. The application data must be written during the clone transaction.

Inputs

adaptor	The adaptor to be cloned.
offset	The location of the adaptor in the application's content.

Outputs

clonedId	The id representing the cloned adaptor.

Errors Returned

kALOEErrCloneNotBegun	A clone transaction has not been started.
kALOEErrInvalidCloneKind	The clone kind specified when the clone was begun is incompatible with this routine.
kALOEInvalidAdaptor	This is not a valid adaptor.

Pre conditions

A clone transaction has been started by calling `ALOEBeginScrapClone` or `ALOEBeginDragClone`.

Post conditions

The returned id represents the cloned adaptor.

See also

`ALOEBeginScrapClone, ALOEBeginDragClone, ALOEPutScrap, ALOEAddDragItemFlavor, ALOECloneAdaptorToContent.`

```
OSErr ALOECloneAdaptorToContent(ALOEClonedID id, ALOEAdaptor* newAdaptor);
```

Basic operation

This routine is used to add a previously cloned adaptor to the application's content. A clone transaction must be in progress. The id should come from application data read via `ALOEGetScrap` (for a scrap clone transaction) or the toolbox routine `GetFlavorData` (for a drag and drop clone transaction). The application data can be read before starting the clone transaction.

If this funtion returns noErr, newAdaptor will be set to a valid adaptor. However, this adaptor should not be passed to other ALOE routines until the clone transaction has been completed by calling ALOEEndClone or ALOEAbortClone. Doing so will result in the error kALOEErrAdaptorNotInitialized. This includes ALOENewAdaptorView, ALOEGetPersistentID, and even ALOEDisposeAdaptor, for example. If the clone transaction is sucessfully completed by calling EndClone, the adaptor may be used as an argument to any ALOE routine. However, if the clone transaction is aborted by calling ALOEAbortClone, (or if ALOEEndClone returns an error), the adaptor can only be passed to ALOEDisposeAdaptor. Other routines will return the error kALOEInvalidAdaptor.

If the newAdaptor value returned is non-null, the application must eventually dispose of the returned adaptor (after the duplicate transaction has been completed) by calling `ALOEDisposeAdaptor`.

Inputs

id	The id of the adaptor previously cloned via `ALOECloneAdaptorFromContent`.

Outputs

newAdaptor	The newly added adaptor.

Errors Returned

kALOEErrCloneNotBegun	A clone transaction has not been started.
kALOEErrInvalidCloneKind	The clone kind specified when the clone was begun is incompatible with this routine.
kALOEErrInvalidClonedID	The id does not represent a cloned adaptor.
memFullErr	Not enough memory available to create adaptor.

Pre conditions

A clone transaction is in progress, and the id represents an adaptor on the ALOE scrap or drag reference.

Post conditions

The returned adaptor is internalized and is a valid argument to ALOE routines.

See also

`ALOEBeginScrapClone, ALOEBeginDragClone, ALOEGetScrap, ALOECloneAdaptorFromContent.`

```
OSErr ALOEAbortClone();
```

Basic operation

Used when the application exchanges data including embedded adaptors. This call is made when the application must abort a clone or drag operation in the middle of a transaction initiated by `ALOEBeginScrapClone`, `ALOEBeginDragClone`, or `ALOEBeginDuplicateClone`. The application may need to abort due to an error from ALOE, or an error generated in the application's code. Any new adaptors returned by `ALOECloneAdaptorToContent` or `ALOEDuplicateAdaptor` during this clone transaction must be immediately disposed by the application. This function completes the cloning transaction; `ALOEEndClone` should not be called.

Inputs

none

Outputs

none

Errors Returned

none

Pre conditions

A clone transaction is in progress.

Post conditions

The clone transaction is terminated. Any adaptors created during the transaction by `ALOECloneAdaptorToContent` or `ALOEDuplicateAdaptor` must be disposed.

See also

`ALOEBeginScrapClone, ALOEBeginDragClone, ALOEBeginDuplicateClone, ALOEEndClone`

```
OSErr ALOEEndClone();
```

Basic operation

Completes the clone transaction in progress. There must be a matching `ALOEEndClone` or `ALOEAbortClone` (in case of error) for each call to `ALOEBeginScrapClone`, `ALOEBeginDragClone` or `ALOEBeginDuplicateClone`.

If a scrap or drag clone was begun, an OpenDoc container is placed on the scrap or drag reference, depending on how the transaction was begun. If no adaptors were cloned, this routine completes the transaction but does not create an OpenDoc container.

If new adaptors were created by `ALOECloneAdaptorToContent` or `ALOEDuplicateAdaptor` during this clone transaction, views on the adaptor may now be created.

Inputs

none

Outputs

none

Errors Returned

none

Pre conditions

`ALOEBeginScrapClone`, `ALOEBeginDragClone` or `ALOEBeginDuplicateClone` has been called to begin a clone transaction

Post conditions

The clone transaction has been completed.

See also

`ALOEBeginScrapClone, ALOEBeginDragClone, ALOEBeginDuplicateClone, ALOEAbortClone`

```
OSErr ALOEEmbedScrap(ALOEAdaptorMgr adaptorMgr, WindowPtr window,
        ALOEAdaptor* newAdaptor);
```

Basic operation

This routineembeds data from the clipboard and returns a new adaptor. The clipboard may contain platform data or an OpenDoc container. This is an atomic routine; an error is returned if this routine is called within a clone transaction begun by `ALOEBeginScrapClone`. The application must eventually dispose of the returned adaptor by calling `ALOEDisposeAdaptor`.

Inputs

adaptorMgr	The manager responsible for the adaptor created by this routine.
window	The window in which to embed.

Outputs

newAdaptor	The newly created adaptor.

Errors Returned

kALOEErrCloneInProgress	A clone transaction is already in progress.
memFullErr	Not enough memory available to create adaptor.

Pre conditions

A clone transaction is not in progress.

Post conditions

The returned adaptor is internalized and is a valid argument to ALOE routines.

Drag and Drop

Drag and Drop uses the same cloning mechanism as the clipboard. Therefore, `ALOECloneAdaptorFromContent`, `ALOECloneAdaptorToContent`, `ALOEEndClone` and `ALOEAbortClone` are used with the Drag and Drop function `ALOEBeginDragClone`.


```
OSErr ALOEBeginDragClone(ALOEAdaptorMgr adaptorMgr, WindowPtr window,
        DragReference theDragRef, ALOECloneKind kind);
```

Basic operation

Used when the application exchanges data including embedded adaptors. This call is used during a drag to begin a transaction involving copying embedded adaptors to a drag reference, and used during a drop to copy adaptors from a drag reference.

When initiating a drag, the application should specify the clone kind kALOEDrag. When receiving a drop, the application should specify kALOEDrop. ALOE will examine the drag modifiers and treat the drop as a copy or a move as appropriate.

There must be a matching `ALOEEndClone` or `ALOEAbortClone` (in case of error) for each call to `ALOEBeginDragClone`.

Inputs

adaptorMgr	The adaptor manager containing adaptors to be cloned during a drag or to created adaptors during a drop.
window	The origin of a drag or destination of a drop.
theDragRef	The drag reference created by the application for the drag or recieved by the application for the drop.
kind	The type of clone (`kALOEDrag` or `kALOEDrop`).

Outputs

long	The function result.

Errors Returned

kALOEErrCloneInProgress	A clone transaction is already in progress.
kALOEInvalidCloneKind	Invalid clone kind.
kALOEInvalidMgr	Adaptor manager is not valid.
Drag Manager Errors	

Pre conditions

A clone transaction is not in progress.

Post conditions

A drag and drop clone transaction has begun.

See also

For interacting with Scrap Manager or clipboard, see `ALOEBeginScrapClone`.

```
OSErr ALOEAddDragItemFlavor(DragReference theDragRef, ItemReference
        theItemRef, FlavorType theType, void* dataPtr, Size dataSize,
        FlavorFlags theFlags);
```

Basic operation

This is a cover routine for the drag manager call `AddDragItemFlavor` and must be used to write application data to a drag reference during a clone transaction started by calling `ALOEBeginDragClone` if embedded adaptors are being cloned. At least one flavor written by the application will usually contain cloned ids returned by `ALOECloneAdaptorFromContent`.

Inputs

theDragRef	The drag reference for the new flavor.
theItemRef	The item for the new flavor data.
theType	The type of the flavor.
dataPtr	The pointer to the data.
dataSize	The size of the data.
theFlags	The flavor flags passed to AddDragItemFlavor.

Outputs

none

Errors Returned

Drag Manager Errors

Pre conditions

A clone transaction begun by `ALOEBeginDragClone` is in progress.

Post conditions

The flavor is added to the drag item.

```
OSErr ALOEEmbedDrag(ALOEAdaptorMgr adaptorMgr, WindowPtr
        window, DragReference theDragRef, long index,
        ALOEAdaptor* newAdaptor);
```

Basic operation

ALOEEmbedDrag embeds data from the indexed drag item of the drag reference as an adaptor view. The drag reference may contain platform data or an OpenDoc container. ALOEEmbedDrag is an atomic routine; an error is returned if this routine is called within an ALOEBeginDragClone transaction. The application must eventually dispose of the returned adaptor by calling ALOEDisposeAdaptor.

Inputs

adaptorMgr	The manager responsible for the adaptor created by this routine.
theDragRef	The drag reference from which to clone the adaptor.
index	The index of the drag item to embed.

Outputs

`newAdaptor`	The new embedded adaptor.

Errors Returned

kALOECannotEmbed	Embedding is not allowed during a drag (i.e., when ALOEBeginDragClone is called.)

Drag Manager Errors

Pre conditions

A clone transaction is not in progress.

Post conditions

The returned adaptor is internalized and is a valid argument to ALOE routines.

Scripting

Initialization

```
OSErr ALOEObjectInit();
```

Basic operation

Container applications must call this routine before calling any of the ALOE routines that describe or manipulate Apple event objects. Container applications should call ALOEObjectInit instead of the toolbox defined AEObjectInit.

Inputs

none

Outputs

none

Errors Returned

noErr	0	No error
memFullErr	-108	Not enough room in heap zone
errAENewerVersion	-1706	Need a newer version of the Apple Event Manager

Pre conditions

none

Post conditions

Container application can safely call ALOE scripting routines.

Event Handlers

```
OSErr ALOEInstallEventHandler(
        AEEventClass theAEEventClass,
        AEEventID theAEEventID,
        AEEventHandlerUPP handler,
        long handlerRefCon,
        Boolean isSysHandler);
```

Basic operation

ALOEInstallEventHandler adds an entry to either your application's Apple event dispatch table or the system Apple event dispatch table. Container applications should call this routine instead of the toolbox-defined AEInstallEventHandler.

Inputs

theAEEventClass	The event class to be dispatched for this entry.
theAEEventID	The event ID to be dispatched for this entry.
handler	A pointer to an Apple event handler for this dispatch table entry.
handlerRefCon	A reference constant passed by ALOE to the handler each time the handler is called.
isSysHandler	If TRUE, the handler is added to the system dispatch table. If FALSE, the handler is added to the application's Apple event manager dispatch table.

Outputs

none

Errors Returned

noErr	0	No error
paramErr	-50	Parameter error (handler pointer is NIL or odd)
memFullErr	-108	Not enough room in heap zone

Pre conditions

none

Post conditions

If the handler was installed in the system's dispatch table (the value of isSysHandler was TRUE) and the address of the handler is in the application's heap, the handler must be removed by calling ALOERemoveEventHandler before the application terminates.

```
OSErr ALOEGetEventHandler(
        AEEventClass theAEEventClass,
        AEEventID theAEEventID,
        AEEventHandlerUPP* handler,
```



```
        long* handlerRefCon,
        Boolean isSysHandler);
```

Basic operation

ALOEGetEventHandler gets an entry from an Apple event dispatch table. Container applications should call this routine instead of the toolbox-defined AEGetEventHandler.

Inputs

theAEEventClass	The value of the event class field of the dispatch table entry for the desired handler.
theAEEventID	The value of the event ID field of the dispatch table entry for the desired handler.
isSysHandler	Specifies the Apple event dispatch table from which to get the handler. If the value of isSysHandler is TRUE, ALOEGetEventHandler returns the handler from the system dispatch table. If isSysHandler is FALSE, ALOEGetEventHandler returns the handler from your application's dispatch table.

Outputs

handler	A pointer to the handler specified by the input parameters.
handlerRefCon	The reference constant associated with the handler specified by the input parameters.

Errors Returned

noErr	0	No error.
errAEHandlerNotFound	-1717	No handler found for the specified event.

Pre conditions

Event handler must be installed.

Post conditions

none

```
OSErr ALOERemoveEventHandler(
        AEEventClass theAEEventClass,
        AEEventID theAEEventID,
        AEEventHandlerUPP handler,
        Boolean isSysHandler);
```

Basic operation

ALOERemoveEventHandler removes an entry from an Apple event dispatch table. Container applications should call this routine instead of the toolbox-defined AERemoveEventHandler.

Inputs

theAEEventClass	The event class field of the handler whose entry should be removed from the dispatch table.
theAEEventID	The event ID of the handler whose entry should be removed from the dispatch table.
handler	A pointer to the handler to be removed. If the value of the parameter is NIL, the handler to be removed is determined solely by the event class and event ID parameters.
isSysHandler	Specifies the dispatch table from which to remove the handler. If the value of isSysHandler is TRUE, ALOERemoveEventHandler removes the specified

handler from the system dispatch table. If the value of isSysHandler is FALSE, ALOE removes the handler from your application's dispatch table.

Outputs

none

Errors Returned

noErr	0	No error
errAEHandlerNotFound	-1717	No handler found for the specified event

Pre conditions

The event handler must be installed.

Post conditions

none

PreDispatch Handler

```
OSErr ALOEInstallPreDispatchHandler(
        AEEventHandlerUPP handler
        Boolean isSysHandler);
```

Basic Operation

ALOEInstallPreDispatchHandler installs a special dispatch handler that is called immediately before the Apple Event Manager dispatches an Apple event. Container applications should call this function instead of the toolbox AEInstallSpecialHandler function. Object method callbacks, which applications have historically been able to install by calling AEInstallSpecialHandler, should be installed by calling ALOESetObjectCallbacks.

Inputs

handler	A pointer to the handler to be called for predispatched Apple events.
isSysHandler	The dispatch table to which to add the handler. If true, the handler is installed in the system's dispatch table. If FALSE, and OpenDoc is installed, the handler is added to ALOE's internal dispatch table. If FALSE, and OpenDoc is not installed, ALOE calls the toolbox AEInstallSpecialHandler with a keyword of keyPreDispatch to install the handler.

Outputs

none

Errors Returned

noErr	0	No error
paramErr	-50	Parameter error (handler pointer is nil or odd)
memFullErr	-108	Not enough room in heap zone

Pre conditions

none

Post conditions

If the handler was installed in the system's dispatch table (the value of isSysHandler was TRUE) and the address of the handler is in the application's heap, the handler must be removed by calling ALOERemoveEventHandler before the application terminates.

Function Prototype

```
OSErr ALOEGetPreDispatchHandler(
      AEEventHandlerUPP* handler
      Boolean isSysHandler);
```


Basic Operation

ALOEGetPreDispatchHandler gets the predispatch handler installed in the specified dispatch table. Container applications should call this routine instead of the toolbox-defined AEGetSpecialHandler.

Inputs

isSysHandler	Specifies the dispatch table from which to get the handler. If the value of isSysHandler is TRUE, the handler is taken from the system's dispatch table. If the value of isSysHandler is FALSE, the handler is taken from the application's dispatch table.

Outputs

handler	A pointer to the specified predispatch handler.

Errors Returned

noErr	0	No error
memFullErr	-108	Not enough room in heap zone

Pre conditions

The predispatch handler must be installed.

Post conditions

none

```
OSErr ALOERemovePreDispatchHandler(
        AEEventHandlerUPP handler
        Boolean isSysHandler);
```

Basic Operation

ALOERemovePreDispatchHandler removes a previously installed PreDispatchHandler. Container applications should call this function instead of the toolbox AERemoveSpecialHandler function.

Inputs

handler	A pointer to the predispatch handler to be removed. If this parameter is NIL, the existing predispatch handler will be removed. If this value is not NIL, the predispatch handler will only be removed if it matches this value.
isSysHandler	Specifies the dispatch table from which to remove the handler. If the value of isSysHandler is TRUE, the handler is removed from the system's dispatch table. If the value of isSysHandler is FALSE, the handler is removed from the application's dispatch table.

Outputs

none

Errors Returned

noErr	0	No error
paramErr	-50	Parameter error (handler pointer is nil or odd)
memFullErr	-108	Not enough room in heap zone

Pre conditions

The predispatch handler must be installed.

Post conditions

none

Object Accessors


```
OSErr ALOEInstallObjectAccessor(
          DescType desiredClass,
          DescType containerType,
          OSLAccessorUPP theAccessor,
          long accessorRefCon,
          Boolean isSysHandler);
```

Basic Operation

ALOEInstallObjectAccessor adds an entry for an object accessor function to either the application's object accessor dispatch table or the system object accessor dispatch table. Container applications should call ALOEInstallObjectAccessor instead of the toolbox-defined AEInstallObjectAccessor.

Inputs

desiredClass	The object class of the Apple event objects to be located by the object accessor function for this table entry.
containerType	The descriptor type of the token used to specify the container for the desired objects. The object accessor function finds objects in containers specified by tokens of this type.
theAccessor	A pointer to the object accessor function for this table entry.
isSysHandler	A value that specifies the object accessor dispatch table to which the entry is added. If the value of isSysHandler is TRUE, ALOE adds the entry to the system object accessor dispatch table. If the value is FALSE, ALOE adds the entry to your application's object accessor dispatch table.

Outputs

none

Errors Returned

noErr	0	No error
paramErr	-50	The accessor pointer is NIL or odd

Pre conditions

ALOEObjectInit must be called prior too calling ALOEInstallObjectAccessor.

Post conditions

If the accessor was installed in the system accessor dispatch table (isSysHandler is TRUE), and the address of the accessor is in the application's heap, the accessor must be removed by calling ALOERemoveObjectAccessor before the application terminates.

```
OSErr ALOEGetObjectAccessor(
          DescType desiredClass,
          DescType containerType,
          OSLAccessorUPP* theAccessor,
          long* accessorRefCon,
          Boolean isSysHandler);
```

Basic Operation

ALOEInstallObjectAccessor gets a pointer to an object accessor function and the value of its reference constant. Container applications should call ALOEGetObjectAccessor instead of the toolbox-defined AEGetObjectAccessor.

Inputs

desiredClass	The object class of the Apple event objects located by

	the requested object accessor function. This parameter can also contain the constant typeWildCard.
containerType	The descriptor type of the token that identifies the container for the objects located by the requested object accessor function. This parameter can also contain the constant typeWildCard.
isSysHandler	A value that specifies the object accessor table from which to get the object accessor function and its reference constant. If the value of isSysHandler is TRUE, ALOEGetObjectAccessor gets the function from the system object accessor dispatch table. If the value of isSysHandler is FALSE, ALOEGetObjectAccessor gets the function from the application's object accessor dispatch table.

Outputs

theAccessor	A pointer to the requested object accessor function.
accessorRefCon	The reference constant from the object accessor dispatch table entry for the specified object accessor function.

Errors Returned

noErr	0	No error
errAEAccessorNotFound	-1723	There is no object accessor function for the specified object class and container type.

Pre conditions

ALOEObjectInit must have been called.
The object accessor must be installed.

Post conditions

none

```
OSErr ALOERemoveObjectAccessor(
        DescType desiredClass,
        DescType containerType,
        OSLAccessorUPP theAccessor,
        Boolean isSysHandler);
```

Basic Operation

ALOERemoveObjectAccessor removes an object accessor function from an object accessor dispatch table. Container applications should call ALOERemoveObjectAccessor instead of the toolbox-defined AERemoveObjectAccessor.

Inputs

desiredClass	The object class of the Apple event objects located by the object accessor function. The desiredClass parameter can also contain the constant typeWildCard.
containerType	The descriptor type of the token that identifies the container for the objects located by the object accessor function. This parameter can also contain the constant typeWildCard.
theAccessor	A pointer to the object accessor function you want to remove. If the value is NIL, the function to remove is determined solely by the values of the desiredClass and containerType parameters.

isSysHandler		A value that specifies the object accessor table from which to remove the object accessor function. If the value of isSysHandler is TRUE, ALOERemoveObjectAccessor removes the function from the system object accessor dispatch table. If the value of isSysHandler is FALSE, ALOERemoveObjectAccessor removes the function from the application's object accessor dispatch table.

Outputs

none

Errors Returned

noErr	0	No error
errAEAccessorNotFound	-1723	There is no object accessor function for the specified object class and container type or the value of theAccessor does not match the value of the installed accessor.

Pre conditions

ALOEObjectInit must have been called.
The object accessor must be installed.

Post conditions

none

Object Callbacks

```
OSErr ALOESetObjectCallbacks(
          OSLCompareUPP compareProc,
          OSLCountUPP countProc,
          OSLDisposeTokenUPP disposeTokenProc,
          OSLGetMarkTokenUPP getMarkTokenProc,
          OSLMarkUPP markProc,
          OSLAdjustMarksUPP adjustMarksProc,
          OSLGetErrDescUPP getErrDescProc);
```

Basic Operation

ALOESetObjectCallbacks sets the object callbacks to be called for your application. Container applications should call ALOESetObjectCallbacks instead of the toolbox-defined AESetObjectCallbacks.

Inputs

compareProc	Either a pointer to the object-comparison function provided by your application or NIL if no function is provided.
countProc	Either a pointer to the object-counting function provided by your application or NIL if no function is provided.
disposeTokenProc	Either a pointer to the token disposal function provided by your application or NIL if no function is provided.
getMarkTokenProc	Either a pointer to the function for returning a mark token provided by your application or NIL if no function is provided.
markProc	Either a pointer to the object-marking function provided by your application or NIL if no function is provided.

adjustMarksProc		Either a pointer to the mark-adjusting function provided by your application or NIL if no function is provided.
getErrDescProc		Either a pointer to the error callback function provided by your application or NIL if no function is provided.

Outputs

none

Errors Returned

noErr	0	No error
paramErr	-50	An invalid handler was provided or ALOEObjectInit was not called
memFullErr	-108	There is not enough room in heap zone

Pre conditions

ALOEObjectInit must have been called.

Post conditions

none

Coercion Handlers

```
OSErr ALOEInstallCoercionHandler(
        DescType fromType,
        DescType toType,
        AECoercionHandlerUPP handler,
        long handlerRefCon,
        Boolean fromTypeIsDesc,
        Boolean isSysHandler);
```

Basic Operation

ALOEInstallCoercionHandler installs a coercion handler routine in either the application or system coercion handler dispatch table.

Inputs

fromType	The descriptor type of the data coerced by the handler.
toType	The descriptor type of the resulting data. If there was already an entry in the specified coercion handler table for the same source descriptor type and result descriptor type, the existing entry is replaced.
handler	A pointer to the coercion handler.
handlerRefCon	A reference constant passed by ALOE to the handler each time the handler is called. If your handler doesn't expect a reference constant, use 0 as the value of this parameter.
fromTypeIsDesc	Specifies the form of the data to be coerced. If the value of this parameter is TRUE, the coercion handler expects the data to be passed as a descriptor record. If the value is FALSE, the coercion handler expects a pointer to the data.
isSysHandler	Specifies the coercion table to which the handler is added. If the value of isSysHandler is TRUE, the handler is added to the system coercion table and made available to all applications. If the value is FALSE, the handler is added to the application coercion table.

Outputs

none

Errors Returned

noErr	0	No error
memFullErr	-108	Not enough room in heap zone

Pre conditions

none

Post conditions

If the handler is installed in the system's coercion table (isSysHandler is TRUE), the application must remove the coercion handler before terminating.

```
OSErr ALOEGetCoercionHandler(
          DescType fromType,
          DescType toType,
          AECoercionHandlerUPP* handler,
          long* handlerRefCon,
          Boolean* fromTypeIsDesc,
          Boolean isSysHandler);
```

Basic Operation

ALOEGetCoercionHandler gets the handler for a specified descriptor type coercion.

Inputs

fromType	The descriptor type of the data coerced by the handler.
toType	The descriptor type of the resulting data.
isSysHandler	Specifies the coercion table from which to get the handler. If the value of isSysHandler is TRUE, the handler is taken from the system coercion table. If the value is FALSE, the handler is taken from the application coercion table.

Outputs

handler	A pointer to the desired coercion handler.
handlerRefCon	The reference constant for the desired handler. ALOE passes this reference constant to the handler each time the handler is called.
fromTypeIsDesc	If ALOEGetCoercionHandler returns a value of TRUE for this parameter, the coercion handler expects the data to be passed as a descriptor record. If the function returns FALSE, the coercion handler expects a pointer to the data.

Errors Returned

noErr	0	No error
memFullErr	-108	Not enough room in heap zone

Pre conditions

The coercion handler must be installed.

Post conditions

none

```
OSErr ALOERemoveCoercionHandler(
          DescType fromType,
          DescType toType,
          AECoercionHandlerUPP handler,
          Boolean isSysHandler);
```


Basic Operation

ALOEGetCoercionHandler gets the handler for a specified descriptor type coercion.

Inputs

fromType	The descriptor type of the data coerced by the handler.
toType	The descriptor type of the resulting data.
handler	A pointer to the coercion handler. If the value of handler is NIL, the handler to be removed is determined solely on the values of fromType and toType.
isSysHandler	Specifies the coercion table from which to get the handler. If the value of isSysHandler is TRUE, the handler is removed from the system coercion table. If the value is FALSE, the handler is removed from the application coercion table.

Outputs

none

Errors Returned

noErr	0	No error
memFullErr	-108	Not enough room in heap zone
errAEHandlerNotFound	-1717	No coercion handler found

Pre conditions

The coercion handler must be installed.

Post conditions

none

```
OSErr ALOEGetCoercionHandler(
          DescType fromType,
          DescType toType,
          AECoercionHandlerUPP* handler,
          long* handlerRefCon,
          Boolean* fromTypeIsDesc,
          Boolean isSysHandler);
```

Basic Operation

ALOEGetCoercionHandler gets the handler for a specified descriptor type coercion.

Inputs

fromType	The descriptor type of the data coerced by the handler.
toType	The descriptor type of the resulting data.
isSysHandler	Specifies the coercion table from which to get the handler. If the value of isSysHandler is TRUE, the handler is taken from the system coercion table. If the value is FALSE, the handler is taken from the application coercion table.

Outputs

handler	A pointer to the desired coercion handler.
handlerRefCon	The reference constant for the desired handler. ALOE passes this reference constant to the handler each time the handler is called.
fromTypeIsDesc	If ALOEGetCoercionHandler returns a value of TRUE for this parameter, the coercion handler expects the data to be passed as a descriptor record. If the function returns FALSE, the coercion handler expects a pointer to the data.

Errors Returned

noErr	0	No error
memFullErr	-108	Not enough room in heap zone

Pre conditions

The coercion handler must be installed.

Post conditions

none

Object Specifier Resolution

```
OSErr ALOEResolve(
        AEDesc* objectSpecifier,
        short callbackFlags,
        AEDesc* theToken,
        ALOEAdaptorView* tokenContext);
```

Basic Operation

ALOEResolve resolves an object specifier record in an Apple event parameter.

Inputs

objectSpecifier	The object specifier record to be resolved
callbackFlags	A value that determines what additional assistance, if any, the application can give the Apple Event Manager when it parses the object specifier record (see Inside Macintosh definition of AEResolve)
theToken	ALOEResolve returns, in this parameter, a token that identifies the Apple Event objects specified by the objectSpecifier parameter
tokenContext	If the token returned by ALOEResolve was created by an ALOE adaptor, ALOEResolve returns the adaptor in the tokenContext parameter. If the token was not created by an adaptor, the value of tokenContext will be NIL

Outputs

none

Errors Returned

noErr	0	No error
paramErr	-50	ALOEObjectInit was not called before this function was called
errAEHandlerNotFound	-1717	The necessary object callback function was not found
errAEImpossibleRange	-1720	The range is not valid because it is impossible for a range to include the first and last objects that were specified
errAEWrongNumberArgs	-1721	The number of operands provided for the kAENot logical operator is not 1
errAEAccessorNotFound	-1723	There is no object accessor function for the specified object class and token descriptor type
errAENoSuchLogical	-1725	The logical operator in a logical descriptor record is not kAEAnd, kAEOr or kAENot
errAEBadTestKey	-1726	The descriptor record in a test key is neither a comparison descriptor nor a logical descriptor

		record
errAENotAnObjectSpec	-1727	The objSpecifier parameter is not an object specifier record
errAENegativeCount	-1729	An object-counting function returned a negative result
errAEEmptyListContainer	-1730	The container for an Apple event object is specified by an empty list

Pre conditions

ALOEObjectInit must have been called.

Post conditions

The container application should attempt to use the theToken parameter only if the value of tokenContext is NIL. A tokenContext value other than NIL indicates that the token was created by a container part and cannot be interpreted by the container application.

Token Disposal

```
OSErr ALOEDisposeToken(
        AEDesc* theToken);
```

Basic Operation

ALOEDisposeToken disposes tokens created during object resolution. Applications must call ALOEDisposeToken instead of the toolbox-defined AEDisposeToken. When a container application calls ALOEDisposeToken, ALOE first determines whether the token being disposed was created by the application or by a contained part. If the token was created by the application, ALOE calls the applications token disposal function if one was provided. If a token disposal function was not provide by the application or if the application's token disposal function returns errAEEventNotHandled, ALOE calls the Apple Event Manager's token disposal function. If the token was created by an embedded part, ALOE calls the part's token disposal function to allow the part to dispose of the token appropriately.

Inputs

theToken	The token to be disposed of.

Outputs

none

Errors Returned

noErr	0	No error

Pre conditions

none

Post conditions

none

Edition Manager Support

```
Boolean ALOEReserveSectionID(long sectionID);
```

Basic operation

This function must be used by applications that support the Edition manager. It allows the application to coordinate the use of Edition manager section ids with ALOE. If the section ID argument is not currently in use by ALOE, it will be reserved for the application's use, and true will be returned. Otherwise, false is returned.

Inputs

sectionID	On input, the section ID the application wishes to use.

Outputs

result	True if the requested section ID was reserved for the application's use

Errors Returned

none

Pre conditions

none

Post conditions

none

Usage Examples

The following examples show what a simple application needs to do to support ALOE. The recipes in this section are not complete.

Starting ALOE

An application can control when ALOE should be loaded. This is done by calling StartALOE. In addition, it should install any relevant callback procs for ALOE. A very simple application may only need to install one or two callback procs. A full-featured application may need to install all of them.

```
OSErr tResult = StartALOE()
ALOESetActivateApplicationCallback (&MyActivateApplication, NULL);
```

Creating an Adaptor Manager

When an application creates a new document, it needs to create an Adaptor Manager for it. This Adaptor Manager works on all the windows associated with the document. Therefore, there is no need to create a new manager for each window opened.

```
ALOEAdaptorMgr newManager = NULL;
OSErr tResult = ALOENewAdaptorMgr(&newManager);
```

Creating an Adaptor

Once the Adaptor Manager is created, ALOE is ready to handle embedding of parts. The following example shows how an application creates a text part within itself. The text part is going to be created with text supplied by the application in the textPtr field. Adaptors can be created many different ways, including: by scrap type, by ISOString, or bound to content (via the clipboard or drag and drop operations)

```
// Create a new text adaptor
ALOEAdaptor newAdaptor;
RgnHandle adaptorShape = ::NewRgn();
::RectRgn(adaptorShape, &myDefaultRect);
ALOEISOString tALOEAdaptorKind = ::ALOEGetKindFromResType('TEXT');
OSErr tResult = ::ALOENewAdaptor(manager,
                        window,
                        tALOEAdaptorKind,
                        textPtr,
                        textSize,
                        adaptorShape,
                        &newAdaptor);
```

Creating an Adaptor View

The Adaptor is mainly used for storage purposes. In order for the Adaptor to render itself, the application must create a View for the Adaptor.

```
ALOEAdaptorView newView;
OSErr tResult = ALOENewAdaptorView(adaptor,
```



```
                              NULL,             // use window port
                              offset,           // offset from window origin
                              NULL,             // not printing view
                              &newView );

      tResult = ALOEDrawAdaptorView(newView, NULL);   // default clipping
```

Deactivating and activating the application

When the application is first launched, it is active. However, when an embedded part is activated by the user, the application becomes inactive. ALOE uses the installed callback to notify the application that it is becoming inactive. The application should remember its active state because it needs this information to determine how it should handle different situations.

When the user clicks on the content area of the document which does not have any part, the application is again activated. ALOE uses the installed callback to notify the application that it is being activated. The application can then change its global flag to reflect the correct state.

```
      // MyActivateApplication which has been installed using
      // ALOESetActivateApplicationCallback (see above).

      void MyActivateApplication (Boolean activate,
                                  EventRecord event,
                                  void *contextPtr)
      {
            gApplicationIsActive = activate;
      }
```

Handling Events

The application needs to work with ALOE to determine the parameters to WaitNextEvent. The reason is that the application no longer possesses all the knowledge of the running process because there are embedded parts in it.

```
      MainEventLoop();
      {
            .
            .
            gotEvent = WaitNextEvent(ALOEGetEventMask(everyEvent),
                                  &event,
                                  ALOEGetSleepTime(GetSleep()),
                                  ALOEGetSleepRegion(myRegion));
            if (gotEvent)
            {
                  // If the event is in the menu bar then we have
                  // To call menu select before we give ALOE a chance
                  // to handle the event
                  short part = FindWindow(event->where, &window);

                  // Save off a copy of the event, since OpenDoc will
                  // modify it in ALOEHandleEvent.
                  EventRecord savedEvent = event;
```



```
            if (part == inMenuBar)
                  MyHandleMenuEvent(event);
            else
            {
                  // Give ALOE first crack at the event
                  Boolean handled = ALOEHandleEvent(event);

                  if (!handled)
                        MyHandleEvent(savedEvent);
            .
            .
            }
      }
```

Handling Update Events

The application should handle the update events. The following code sample shows what the application should do:

```
case updateEvt:
      BeginUpdate(windowPtr);
      MyDrawContent();

      while ((myAdaptorView = myAdaptorViewList->Next()) != nil)
      {
            OSErr tResult;
            tResult = ALOEDrawAdaptorView(myAdaptorView, clipRgn);
      }

      EndUpdate(windowPtr);
break;
```

Adjusting menus

The application needs to handle the mouse down event in the menu. The following code sample shows what an application might do in response to a mouse down in the menu bar.

```
//If this is an event in the MenuBar, adjustMenus BEFORE OpenDoc.

if(EventUpdatesMenus(&event))
      MyAdjustMenus();
```

The following is the EventUpdatesMenu function:

```
Boolean EventUpdatesMenus(EventRecord* theEvent)
{
      Boolean updateMenus = false;
      WindowPtr theWindow = NULL;
      short partCode = ::FindWindow(theEvent->where, &theWindow);

      if (partCode == inMenuBar)
            updateMenus = true;
```



```
        return updateMenus;
    }
```

The following shows the menu handling of the application's adjust menu procedure. The cases shown are ALOE-related. The rest of the procedure is application-specific and does not need to be modified for ALOE.

```
    void MyAdjustMenus()
    {
        .
        .
        // Handle undo menu item
        Boolean ALOEHasUndo = ALOEHasPendingUndo();
        if (!ALOEHasUndo && gApplicationHasUndoAction)
            MyEnableCommand(cUndo, true);

        // Handle Save item
        Boolean changed;
        Boolean sameSize;
        OSErr tResult = ALOEIsAdaptorMgrDirty(manager,
                                              &changed,
                                              &sameSize);
        if (changed && gIHaveChanged)
            MyEnableCommand(cSave, true);
        .
        .
    }
```

Saving the document

When an application needs to save a document, it needs to iterate through all the Adaptors and acquire persistent references to them. Then it can store these references in its data fork together with its content. Finally, it needs to write out the storage for the Adaptor Manager. The storage of the Adaptor Manager contains all the embedded parts and the mapping of persistent references to the embedded parts.

```
    // Opening the file and get the refNum
    refNum = ...

    // Iterate through my adaptors in this document
    while ((myAdaptor = myAdaptorList->Next()) != NULL)
    {
        // get the persistent reference
        ALOEPersistentID id;
        OSErr err = ALOEGetPersistentID( myAdaptor, &id );

        // Write out the id in my content with related information,
        // i.e. geometry.
        ......
    }

    // Write out the storage for Adaptor Manager.
    long bytesWritten;
```



```
OSErr tResult = ALOEWriteAdaptorMgrToFile(manager,
                                          refNum,
                                          &bytesWritten);
```

Opening a document

Opening a document is the reverse process of saving a document. The application needs to create the Adaptor Manager which has all the information of the embedded parts.

```
// Opening the file and get the refNum
refNum = ...

// Create the Adaptor Manager from a file.

long bytesWritten;
ALOEAdaptorMgr manager = NULL;
OSErr err = ALOENewAdaptorMgrFromFile (refNum, &manager);
...

// Going through the data fork. When the application sees
// a persistent id, it can get the adaptor back using the id.

ALOEAdaptor newAdaptor;
err = ALOEGetAdaptor( manager, id, window, &newAdaptor );
...
```

Terminating ALOE

The application should dispose of all the Views, Adaptors and the Adaptor Managers it has created when it is shutting itself down.

```
OSErr err;
err = ALOEDisposeAdaptorView(view);
err = ALOEDisposeAdaptor(adaptor);
err = ALOEDisposeAdaptorMgr(manager);
err = TerminateALOE();
```

ALOE Clipboard

Copy - Application content only.

When writing application content only, the ALOE scrap manager cover routines are used.

```
err = ALOEZeroScrap();
err = ALOEPutScrap(length, 'TYPE', dataPtr);
```

Copy - One embedded adaptor.

Embedded adaptors are copied in a transaction begun with `ALOEBeginScrapClone` and ended by ALOEEndClone. When exactly one embedded adaptor is copied, the application does not call `ALOEPutScrap` to put an application kind on the scrap.

Typically, the offset argument to `ALOECloneAdaptorFromContent` is the offset of the adaptor view being copied.

```
ALOEClonedID theClonedID;

err = ALOEZeroScrap();
err = ALOEBeginScrapClone(theAdaptorMgr, theWindow, kALOECopy);
err = ALOECloneAdaptorFromContent(theAdaptor, theOffset, theClonedID);
err = ALOEEndClone();
```

Copy - Hybrid content.

When more than one embedded adaptor is copied, or when application content and adaptors are copied, the application clones the adaptors and writes one or more scrap types within a clone transaction. When no application content is copied, the scrap data written by the application is just a "wrapper" containing the cloned ids returned by `ALOECloneAdaptorFromContent`. Besides the cloned ids, this wrapper typically contains whatever positional information the application associates with the cloned adaptors.

```
ALOEClonedID oneClonedID, twoClonedID;

err = ALOEZeroScrap();
err = ALOEBeginScrapClone(theAdaptorMgr, theWindow, kALOECopy);

err = ALOECloneAdaptorFromContent(oneAdaptor, oneOffset, oneClonedID);
err = ALOECloneAdaptorFromContent(twoAdaptor, twoOffset, twoClonedID);

// The data of this type includes oneClonedID & twoClonedID
err = ALOEPutScrap(length, theHybridType, dataPtr);

err = ALOEEndClone();
```

Copy - Promising data.

If your application maintains its own private scrap, your part can postpone, and perhaps avoid, the expense of converting your private scrap to scrap data by writing a "promise' instead of actual data. Your application writes a promise by passing null as the source pointer argument to ALOEPutScrap. If ALOE needs the actual data, because the user performed a paste in an embedded adaptor, or because the process was suspended, ALOE will request that data by calling your application's WriteScrap callback. You must have previously installed a WriteScrap callback before calling ALOEPutScrap to promise data.

Your WriteScrap callback routine must have the following signature:

```
OSErr WriteScrapProc(ResType theType, void *contextPtr);
```

Your WriteScrap callback will be called to fulfill a promise for the argument scrap type. Your callback does not need to fulfill promises for other scrap types, although it may.

When the copied content includes one or more embedded adaptors, your application should clone the adaptors immediately, but may promise the (hybrid) scrap type in your call to

ALOEPutScrap. See the example for copying hybrid content.

Enabling the Paste menu item.

Since any content can be embedded, this menu should be enabled unless the scrap is empty.

```
if ( ALOEIsScrapEmpty() )
      DisableItem(editMenu, thePasteItem);
else
      EnableItem(editMenu, thePasteItem);
```

Paste - Deciding which scrap data to accept.

How your application pasts data from the scrap depends on which scrap type it chooses to paste. Generally, your application should look for a scrap type it can read. If no such type is found, just embed the scrap.

```
// Check for hybrid content first
if ( ALOEGetScrap(NULL, theHybridType, &offset) > 0 )
      Paste hybrid content
else if ( ALOEGetScrap(NULL, thePlainType, &offset) > 0 )
      Incorporate application data
else
      Embed the scrap
```

Paste - Incorporating application data.

Application data is incorporated using ALOE's GetScrap cover routine.

```
err = ALOEGetScrap(dataHandle, thePlainType, &offset);
```

Paste - Embedding the scrap.

The contents of the scrap can be embedded as an adaptor using an atomic routine.

```
ALOEAdaptor newAdaptor;
err = ALOEEmbedScrap(theAdaptorMgr, theWindow, &newAdaptor);

ALOEAdaptorView newView;
err = ALOENewAdaptorView(newAdaptor, nullGrafPtr, theOffset,
                         nullTHPrint, &newView);
```

Paste - Hybrid content.

To paste hybrid content, the application reads its scrap type and clones each adaptor into its content.

To make pasted adaptors are visible, views are created AFTER ALOEEndClone has been cloned.

Note that the adaptor offset argument supplied to ALOECloneAdaptorFromContent is not returned by ALOECloneAdaptorToContent. Your application should keep whatever positional information it needs for a cloned adaptor in the scrap data it writes.

```
err = ALOEBeginScrapClone(theAdaptorMgr, theWindow, kALOEPaste);

// Read the application's content, including the cloned ids of adaptors
err = ALOEGetScrap(dataHandle, theHybridType,  &offset);

// Clone each embedded adaptor
ALOEAdaptor oneAdaptor, twoAdaptor;
err = ALOECloneAdaptorToContent(oneID, &oneAdaptor);
err = ALOECloneAdaptorToContent (twoID, &twoAdaptor);

err = ALOEEndClone();

// If the adaptors are visible, show them by creating views
ALOEAdaptorView oneView, twoView;
err = ALOENewAdaptorView(oneAdaptor, nullGrafPtr, theOffset,
                         nullTHPrint, &oneView);

err = ALOENewAdaptorView(twoAdaptor, nullGrafPtr, theOffset,
                         nullTHPrint, &twoView);
```

Scrap coordination.

If your application maintains a private scrap, you need to coordinate your private scrap with ALOE's scrap and with the desktop scrap.

When your private scrap changes, notify ALOE by calling ALOEZeroScrap, and by calling ALOEPutScrap for each scrap type available. To avoid the expense of converting your private scrap to scrap data, you can pass null as the source pointer argument to ALOEPutScrap. This represents a "promise" for the data, which ALOE can latter request via your application's WriteScrap callback. You must have previously installed a WriteScrap callback to promise data.

After you applications calls ALOEZeroScrap, call ALOEGetScrapCount to keep track of the scrap generation associated with your change. Your private scrap is valid as long as the scrap's generation doesn't change.

```
err = ALOEZeroScrap();
err = ALOEPutScrap(0, theScrapType, NULL);  // Repeat for each type
scrapCount = ALOEGetScrapCount();
```

Before using data from your private scrap, make sure the scrap generation hasn't changed by calling ALOEScrapCountStillValid. If this routine returns false, read data from the ALOE scrap instead.

```
if ( ALOEScrapCountStillValid(scrapCount) )
{
      // use your application's private scrap
}
```



```
else
{
      // use ALOE's scrap routines
      err = ALOEGetScrap(dataHandle, theHType,  &offset);
}
```

Your application can also use ALOEScrapCountStillValid to decide when its safe to discard your private scrap.

If your application uses a toolbox routine that writes directly to the desk scrap, such as TextEdit, you must NOT use ALOE's scrap cover routines. Instead, use the scrap manager routines directly. The next time data is requested, ALOE will notice that the desk scrap contains the most recent data. For example, to transfer data from a TextEdit record to the scrap, use the following sequence of calls:

```
err = ZeroScrap();      // Calling ALOEZeroScrap would confuse ALOE!
err = TEToScrap();
```

Before your application calls a routine that reads the desktop scrap directly, such as TextEdit, call ALOEUpdateScrap to ensure the latest data is on the desktop scrap. ALOEUpdateScrap will transfer all scrap types to the desktop scrap. (OpenDoc content is NOT placed on the scrap by this call to avoid the unnecessary expense of externalizing the OpenDoc clipboard. However, if text was copied in an OpenDoc part, for example, the text representation will be transferred to the desktop scrap.)

```
err = ALOEUpdateScrap();
```

ALOE Drag and Drop

Dragging Hybrid Content.

This example demonstrates how to initiate a drag of application data and embedded adaptors. Note the similarity of this example to that of copying hybrid content to the clipboard.

```
DragReference theDragRef;

err = NewDrag(&theDragRef);

ALOEClonedID oneClonedID, twoClonedID;

err = ALOEBeginDragClone(theAdaptorMgr, theWindow, kALOEDrag);

err = ALOECloneAdaptorFromContent(oneAdaptor, oneOffset, oneClonedID);
err = ALOECloneAdaptorFromContent(twoAdaptor, twoOffset, twoClonedID);

// The data of this type includes oneClonedID & twoClonedID
err = ALOEAddDragItemFlavor(theDragRef, 1, theHybridType, dataPtr,
                                 length, theFlags);
err = ALOEEndClone();

// Optional but recommended - set the bounding rectangle for the
// dragged content.
```



```
err = SetDragItemBounds(theDragRef, 1, &itemBoundsRect);

// Start the drag
err = TrackDrag(theDragRef, event, dragRgn);

// If the content was moved, delete it from the application.
// Your application should also delete the content if it was
// dropped to the trash.
```

Determining if a drop was a move or a copy.

After StartDrag returns noErr, your application can use code similar to the following to determine if the drop was a copy or a move .

```
ODSShort mseDownMods;
ODSShort mseUpMods;
Boolean isMove = false;

GetDragModifiers(theDragRef, 0L, &mseDownMods, &mseUpMods);

// Check the mandatory modifier keys first
if ( !(mseUpMods & optionKey) && (mseUpMods & controlKey) )
{
      isMove = true;
}

// Then check the optional modifier keys

else if ( !(mseDownMods & optionKey) && (mseDownMods & controlKey) )
{
      isMove = true;
}
```

