INSIDE MACINTOSH

Mac OS 8
Window Manager Reference

Updated for Appearance 1.0.2

November 11, 1998
Technical Publications

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

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

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.

Contents

Figures, Tables, and Listings

CHAPTER 1

Mac OS 8 Window Manager Reference

Contents

The Window Manager defines a set of standard windows and provides functions for creating, displaying, and managing their drawing and behavior. Applications use windows for most communication with the user, from discrete interactions such as presenting and acknowledging alert boxes to open-ended interactions such as creating and editing documents. A window can be any size or shape, and the user can display any number of windows, within the limits of available memory, on the screen at once.

Portions of the Window Manager application programming interface (API) are new, changed, or not recommended with Mac OS 8 or Appearance Manager 1.0. See the following sections for descriptions of the changes to the Window Manager:

- “Window Manager Functions” (page 9)
- “Window Manager Data Types” (page 28)
- “Window Manager Constants” (page 32)

For descriptions of the parts of the Window Manager API that are unaffected by Mac OS 8 or Appearance Manager 1.0, see *Inside Macintosh: Macintosh Toolbox Essentials*. For a description of the Mac OS 8.5 Window Manager API, see *Mac OS 8.5 Window Manager Reference*.

Window Manager Functions

Window Manager functions in the following areas have been affected by Mac OS 8 or Appearance Manager 1.0:

- “Creating Windows” (page 10)
- “Retrieving Window Information” (page 10)
- “Displaying Windows” (page 13)
- “Collapsing Windows” (page 14)
- “Manipulating Window Color Information” (page 17)
- “Defining Your Own Window Definition Function” (page 18)

Creating Windows

The following Window Manager functions for creating windows are not recommended with Mac OS 8:

- `GetNewWindow` (page 10) creates a new monochrome window from a window resource. Not recommended with Mac OS 8.
- `NewWindow` (page 10) creates a new monochrome window from a parameter list. Not recommended with Mac OS 8.

GetNewWindow

Creates a new monochrome window from a window resource. The `GetNewWindow` function was originally implemented prior to Color QuickDraw. In Mac OS 8, you should call the Color QuickDraw function `GetNewCWindow` instead of `GetNewWindow` to programmatically create a window, because Color QuickDraw is always available in Mac OS 8.

VERSION NOTES

Not recommended with Mac OS 8 and later.

NewWindow

Creates a new monochrome window from a parameter list. The `NewWindow` function was originally implemented prior to Color QuickDraw. In Mac OS 8, you should call the Color QuickDraw function `NewCWindow` instead of `NewWindow` to programmatically create a window, because Color QuickDraw is always available in Mac OS 8.

VERSION NOTES

Not recommended with Mac OS 8 and later.

Retrieving Window Information

The following Window Manager functions for retrieving window information are new or changed with Appearance Manager 1.0:

- `FindWindow` (page 11) maps the location of the cursor to a part of the screen or a region of a window when your application receives a mouse-down event. Changed with Appearance Manager 1.0.
- `GetWindowFeatures` (page 12) obtains the features that a window supports. New with Appearance Manager 1.0.
- `GetWindowRegion` (page 12) obtains a handle to a specific window region. New with Appearance Manager 1.0.

FindWindow

Maps the location of the cursor to a part of the screen or a region of a window when your application receives a mouse-down event.

```
pascal short FindWindow (
                    Point thePoint,
                    WindowPtr *theWindow);
```

`thePoint` The point, in global coordinates, where the mouse-down event occurred. Your application retrieves this information from the `where` field of the event structure.

`theWindow` A pointer to the window in which the mouse-down event occurred. `FindWindow` produces `nil` if the mouse-down event occurred outside a window.

function result A short integer that specifies the location of the cursor when the user pressed the mouse button; see "FindWindow Result Code Constants" (page 46).

DISCUSSION

You typically call the function `FindWindow` whenever you receive a mouse-down event. The `FindWindow` function helps you dispatch the event by reporting whether the cursor was in the menu bar or in a window when the mouse button was pressed. If the cursor was in a window, the function will produce both a pointer to the window and a constant that identifies the region of the window in which the event occurred. If Appearance is available, `FindWindow` may return the `inCollapseBox` constant as one of the possible window regions.

VERSION NOTES

Changed with Appearance Manager 1.0 to support finding events in the collapse box.

GetWindowFeatures

Obtains the features that a window supports.

```
pascal OSStatus GetWindowFeatures (
                    WindowPtr inWindow,
                    UInt32* outFeatures);
```

`inWindow` A pointer to the window to be examined.

`outFeatures` A pointer to an unsigned 32-bit value. On return, the bits of the value specify the features the window supports; see "Reporting Window Features" (page 26).

function result A result code; see "Result Codes" (page 48).

DISCUSSION

The `GetWindowFeatures` function produces a window definition function's features in response to a `kWindowMsgGetFeatures` message. For a list of the features a window might support, see "Reporting Window Features" (page 26).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

GetWindowRegion

Obtains a handle to a specific window region.

```
pascal OSStatus GetWindowRegion (
                    WindowPtr inWindow,
                    WindowRegionCode inRegionCode,
                    RgnHandle ioWinRgn);
```

`inWindow`	A pointer to the window to be examined.
`inRegionCode`	A constant representing the window region whose handle you wish to obtain; see "Window Region Constants" (page 44).
`ioWinRgn`	A handle to a region created by your application. On return, the handle is set to the specified window region.
function result	A result code; see "Result Codes" (page 48).

DISCUSSION

The `GetWindowRegion` function produces a handle to a window definition function's window region in response to a `kWindowMsgGetRegion` message. The visibility of the window is unimportant for `GetWindowRegion` to work correctly.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Displaying Windows

The following Window Manager function for displaying windows is changed with Appearance Manager 1.0:

- `DrawGrowIcon` (page 13) draws a window's size box and scroll bar delimiters. Changed with Appearance Manager 1.0.

DrawGrowIcon

Draws a window's size box and scroll bar delimiters.

```
pascal void DrawGrowIcon (WindowPtr theWindow);
```

`theWindow`	A pointer to a window.

DISCUSSION

The `DrawGrowIcon` function draws a window's size box or, if the window can't be resized, whatever other image is appropriate to the window's type and active/

inactive state. When you adopt the Appearance Manager directly, you never need to call the `DrawGrowIcon` function to get a size box in your window. Although you can still use `DrawGrowIcon` to draw the delimiting scroll bar lines, if you wish.

If you are going through the mapping layer, however, you do need to call `DrawGrowIcon`, but only once. This is because, with the Appearance Manager, once `DrawGrowIcon` is called, the size box is merged from the content region of the window into the window's frame.

Note that the `DrawGrowIcon` function doesn't erase the scroll bar areas. If you use `DrawGrowIcon` to draw the size box and scroll bar outline, therefore, you should erase those areas yourself when the window size changes, even if the window doesn't contain scroll bars.

For inactive document windows, `DrawGrowIcon` draws the lines delimiting the size box and scroll bar areas and erases the size box. For active document windows, `DrawGrowIcon` draws the grow image in the size box in the lower-right corner of the window's graphics port rectangle, along with the lines delimiting the size box and scroll bar areas. To draw the size box but not the scroll bar outline, set the `clipRgn` field in the window's graphics port to be a 15-by-15 pixel rectangle in the lower-right corner of the window.

VERSION NOTES

Usage changed with Appearance Manager 1.0; you need call `DrawGrowIcon` only when not adopting the Appearance Manager directly.

Collapsing Windows

The following Window Manager functions for collapsing windows are new with Appearance Manager 1.0:

- `CollapseWindow` (page 15) collapses or expands a window to its title bar. New with Appearance Manager 1.0.
- `CollapseAllWindows` (page 15) collapses or expands all collapsable windows in an application. New with Appearance Manager 1.0.
- `IsWindowCollapsed` (page 16) determines whether a window is currently collapsed. New with Appearance Manager 1.0.

- `IsWindowCollapsable` (page 17) determines whether a window can be collapsed. New with Appearance Manager 1.0.

CollapseWindow

Collapses or expands a window to its title bar .

```
pascal OSStatus CollapseWindow (
                    WindowPtr inWindow,
                    Boolean inCollapseIt);
```

`inWindow` A pointer to a window.

`inCollapseIt` A Boolean value indicating whether the window should be collapsed or expanded.

function result A result code; see "Result Codes" (page 48).

DISCUSSION

The `CollapseWindow` function tells the Window Manager to collapse or expand a window depending upon the value passed in the `inCollapseIt` parameter. Only window definition functions that return the feature bit `kWindowCanCollapse` in response to a `kWindowGetFeatures` message support this function; see `GetWindowFeatures` (page 12).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

CollapseAllWindows

Collapses or expands all collapsable windows in an application.

```
pascal OSStatus CollapseAllWindows (Boolean inCollapseEm);
```

`inCollapseEm` A Boolean value. Set to `true` to collapse all windows in the application. Set to `false` to expand all windows in the application.

function result A result code; see “Result Codes” (page 48).

DISCUSSION

Only window definition functions that return the feature bit `kWindowCanCollapse` in response to a `kWindowGetFeatures` message support the `CollapseAllWindows` function; see `GetWindowFeatures` (page 12).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

IsWindowCollapsed

Determines whether a window is currently collapsed.

```
pascal Boolean IsWindowCollapsed (WindowPtr inWindow);
```

`inWindow` A pointer to the window to be examined.

function result A Boolean value. If `true`, the window is collapsed. If `false`, the window is expanded.

DISCUSSION

Only window definition functions that return the feature bit `kWindowCanCollapse` in response to a `kWindowGetFeatures` message support this function; see `GetWindowFeatures` (page 12). Your window definition function should call `IsWindowCollapsed` to determine whether or not a window is collapsed, so you can modify its structure and content regions as appropriate. Typically, a window’s content region is empty in a collapsed state.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

IsWindowCollapsable

Determines whether a window can be collapsed.

```
pascal Boolean IsWindowCollapsable (WindowPtr inWindow);
```

`inWindow` A pointer to the window to be examined.

function result A Boolean value. If `true`, the window can be collapsed; otherwise, `false`.

DISCUSSION

Your program can call the `IsWindowCollapsable` function to determine if a given window can be programmatically collapsed by calling the function `CollapseWindow` (page 15). Whether a window is collapsable is determined by whether the window definition function returns the feature bit `kWindowCanCollapse` in response to a `kWindowGetFeatures` message.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Manipulating Window Color Information

The following Window Manager functions for manipulating window color information are not recommended with Appearance Manager 1.0:

- `GetAuxWin` (page 17) obtains a handle to a window's auxiliary window structure. Not recommended with Appearance Manager 1.0.
- `SetWinColor` (page 18) sets a window color table. Not recommended with Appearance Manager 1.0.

GetAuxWin

Obtains a handle to a window's auxiliary window structure. When the Appearance Manager is available and you are using standard windows, most of the fields of the auxiliary window structure are ignored and the `GetAuxWin` function is not recommended. In the future, standard system windows may not

have entries in the auxiliary window list. If you are creating your own window definition function, `GetAuxWin` can still be used.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

SetWinColor

Sets a window color table. When the Appearance Manager is available and your application uses standard windows, the `SetWinColor` function is not recommended. `SetWinColor` sets a window color table structure that is now mostly ignored. Only the part identifier constant `wContentColor` in the `value` field of the `ColorSpec` structure is used. This constant produces the background color for the window's content region. Under Appearance, instead of the `SetWinColor` function, the window definition function determines what colors it will use.

If your are creating your own custom window definition functions, `SetWinColor` can still be used.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

Defining Your Own Window Definition Function

The following Window Manager application-defined function is changed with Appearance Manager 1.0:

- `MyWindowDefProc` (page 18) defines a custom window. Changed with Appearance Manager 1.0.

MyWindowDefProc

Application-defined window definition functions are changed with Appearance Manager 1.0 to support collapse boxes and feature reporting. See "Window

Definition Message Constants” (page 21) and “Reporting the Region of a Mouse-Down Event” (page 23) for descriptions of these changes.

A window definition function determines how a window looks and behaves. Various Window Manager functions call a window definition function whenever they need to perform a window-dependent action, such as drawing the window on the screen. If you wish to define new, nonstandard windows for your application, you must write a window definition function and store it in a resource file as a resource of type `'WDEF'`.

The Window Manager calls the Resource Manager to access your window definition function with the given resource ID; see “Window Definition IDs” (page 32) for a description of how window definition IDs are derived from resource IDs and variation codes. You can define your own window variation codes so that you can use one `'WDEF'` resource to handle several variations of the same general window.

The Resource Manager reads your window definition function into memory and returns a handle to it. The Window Manager stores this handle in the `windowDefProc` field of the window structure. Later, when it needs to perform an action on the window, the Window Manager calls the window definition function and passes it the variation code as a parameter.

Your window definition function is responsible for

- drawing the window frame
- reporting the region where mouse-down events occur
- calculating the window’s structure region and content region
- drawing the size box
- resizing the window frame when the user drags the size box
- reporting the window’s features or the location of a specific window region
- performing any customized initialization or disposal tasks

The Window Manager declares the type for an application-defined window definition function as follows:

```
typedef pascal long (*WindowDefProcPtr)(
                     short varCode,
                     WindowPtr theWindow,
                     short message,
                     long param);
```

The Window Manager defines the data type `WindowDefUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr WindowDefUPP;
```

You typically use the `NewWindowDefProc` macro like this:

```
WindowDefUPP myWindowDefUPP;
myWindowDefUPP = NewWindowDefProc(MyWindow);
```

You typically use the `CallWindowDefProc` macro like this:

```
CallWindowDefProc(myWindowDefUPP, varCode, theWindow, message, param);
```

Here's how to declare the function `MyWindowDefProc`:

```
pascal long MyWindowDef(
                    short varCode,
                    WindowPtr theWindow,
                    short message,
                    long param);
```

`varCode` The window's variation code.

`theWindow` A pointer to the window's window structure.

`message` A value indicating the task to be performed. The `message` parameter contains one of the values defined in "Window Definition Message Constants" (page 21). The subsections that follow explain each of these tasks in detail.

`param` Data associated with the task specified by the `message` parameter. If the task requires no data, this parameter is ignored.

function result Your window definition function should perform whatever task is specified by the `message` parameter and return a function result, if appropriate. If the task performed requires no result code, return 0.

Window Definition Message Constants

The Window Manager passes a value defined by one of these constants in the `message` parameter of your window definition function to specify the action your function must perform. Other messages are reserved for internal use by the system.

```
enum {
    wDraw                   = 0,
    wHit                    = 1,
    wCalcRgns               = 2,
    wNew                    = 3,
    wDispose                = 4,
    wGrow                   = 5,
    wDrawGIcon              = 6,
    kWindowMsgGetFeatures   = 7,
    kWindowMsgGetRegion     = 8
};
```

Constant descriptions

`wDraw` Draw the window's frame.

`wHit` Report the location of a mouse-down event.

`wCalcRgns` Calculate the structure region and the content region.

`wNew` Perform additional initialization.

`wDispose` Perform additional disposal.

`wGrow` Draw the dotted outline of the window that you see during a resizing operation.

`wDrawGIcon` Draw the outlines for the size box and the scroll bar.

`kWindowMsgGetFeatures`
Report the window's features. Available with Appearance Manager 1.0 and later.

`kWindowMsgGetRegion`
Report the location of a specific window region. Available with Appearance Manager 1.0 and later.

Drawing the Window Frame

When the Window Manager passes `wDraw` in the `message` parameter, your window definition function should respond by drawing the window frame in

the current graphics port (which is the Window Manager port). The window part code to be drawn will be passed in the `param` parameter of your window definition function.

Your window definition function should perform the following steps:

- Change the current port from the `WMgrPort` to the `WMgrCPort` to allow the system to draw in the full range of RGB colors.
- Update the pen attributes, text attributes, and `bkPat` fields in the `WMgrCPort` to the values of the corresponding fields in the `WMgrPort`. The Window Manager automatically transfers the `vis` and `clip` regions.

Note
The parallelism of the `WMgrPort` and the `WMgrCPort` is maintained only by the window definition functions. All window definition functions that draw in the `WMgrPort` should follow the steps listed above even if the changed fields do not affect their operation.

You must make certain checks to determine exactly how to draw the frame. If the value of the `visible` field in the window structure is `false`, you should do nothing; otherwise, you should examine the `param` parameter and the status flags in the window structure:

- If the value of `param` is 0, draw the entire window frame (including the size box, if your window definition function incorporates the size box into the frame).
- If the value of `param` is 0 and the `hilited` field in the window structure is `true`, highlight the frame to show that the window is active.
  - If the value of the `goAwayFlag` field in the window structure is also `true`, draw a close box in the window frame.
  - If the value of the `spareFlag` field in the window structure is also `true`, draw a zoom box in the window frame.
- If the value of the `param` parameter is `wInGoAway`, add highlighting to, or remove it from, the window's close box.
- If the value of the `param` parameter is `wInZoom`, add highlighting to, or remove it from, the window's zoom box.
- If the value of the `param` parameter is `wInCollapseBox`, add highlighting to, or remove it from, the window's collapse box.

You need to maintain your own state flag to determine whether the close, zoom, or collapse box is to be drawn as highlighted. Typically, you clear this state flag whenever you draw the entire frame, and you set it before drawing whenever your application is called to draw just the close, zoom, or collapse box. If the flag is set, you draw the box in a highlighted state.

The window frame typically, but not necessarily, includes the window's title, which should be displayed in the system font and system font size. The Window Manager port is already set to use the system font and system font size.

Note
Nothing drawn outside the window's structure region will be visible.

Your window definition function should return 0 as the function result for this message.

Reporting the Region of a Mouse-Down Event

When the Window Manager passes `wHit` in the `message` parameter, your window definition function should respond by reporting the region of the specified mouse-down event. The mouse location (in global coordinates) of the window frame will be passed into the `param` parameter of your window definition function. The vertical coordinate is in the high-order word of the parameter, and the horizontal coordinate is in the low-order word.

In response to the `wHit` message, your window definition function should return one of the following constants:

```
enum {
    wNoHit          = 0,
    wInContent      = 1,
    wInDrag         = 2,
    wInGrow         = 3,
    wInGoAway       = 4,
    wInZoomIn       = 5,
    wInZoomOut      = 6,
    wInCollapseBox  = 9
};
```

Constant descriptions

`wNoHit` The mouse-down event did not occur in the content region or the drag region of any active or inactive window or in the close, size, zoom, or collapse box of an active window. The return value `wNoHit` might also mean that the point isn't in the window. The standard window definition functions, for example, return `wNoHit` if the point is in the window frame but not in the title bar.

`wInContent` The mouse-down event occurred in the content region of an active or inactive window (with the exception of the size box).

`wInDrag` The mouse-down event occurred in the drag region of an active or inactive window.

`wInGrow` The mouse-down occurred in the size box of an active window.

`wInGoAway` The mouse-down event occurred in the close box of an active window.

`wInZoomIn` The mouse-down event occurred in the zoom box of an active window that is currently in the standard state.

`wInZoomOut` The mouse-down event occurred in the zoom box of an active window that is currently in the user state.

`wInCollapseBox` The mouse-down event occurred in the collapse box of an active window. Available with Appearance Manager 1.0 and later.

Return the constants `wInGrow`, `wInGoAway`, `wInZoomIn`, `wInZoomOut`, and `wInCollapseBox` only if the window is active—by convention, the size box, close box, zoom box, and collapse box aren't drawn if the window is inactive. In an inactive document window, for example, a mouse-down event in the part of the title bar that would contain the close box if the window were active is reported as `wInDrag`.

Calculating Regions

When the Window Manager passes `wCalcRgns` in the `message` parameter, your window definition function should respond by calculating the window's structure and content regions based on the current graphics port's port rectangle. These regions, whose handles are in the `strucRgn` and `contRgn` fields of the window structure, are in global coordinates. The Window Manager requests this operation only if the window is visible. The mouse location (in

global coordinates) of the window frame will be passed into the `param` parameter of your window definition function.

Your window definition function should call `IsWindowCollapsed` (page 16) to determine its collapse state. Then your window definition function can modify its structure and content regions as appropriate. Typically, a window's content region is empty in a collapsed state.

▲ **WARNING**
When you calculate regions for your own type of window, do not alter the clip region or the visible region of the Window Manager port. The Window Manager and QuickDraw take care of this for you. Altering the Window Manager port's clip region or visible region may damage other windows.

Your window definition function should return 0 as the function result for this message.

Performing Additional Window Initialization

When the Window Manager passes `wNew` in the `message` parameter, your window definition function should respond by performing any initialization that it may require. If the content region has an unusual shape, for example, you might allocate memory for the region and store the region handle in the `dataHandle` field of the window structure. The initialization function for a standard document window creates the `wStateData` structure for storing zooming data.

Your window definition function should ignore the `param` parameter and return 0 as the function result for this message.

Performing Additional Window Disposal Actions

When the Window Manager passes `wDispose` in the `message` parameter, your window definition function should respond by performing any additional tasks necessary for disposing of a window. You might, for example, release memory that was allocated by the initialization function. The dispose function for a standard document window disposes of the `wStateData` structure.

Your window definition function should ignore the `param` parameter and return 0 as the function result for this message.

Drawing the Window's Grow Image

When the Window Manager passes `wGrow` in the `message` parameter, your window definition function should respond to being resized by drawing a dotted outline of the window in the current graphics port in the pen pattern and mode. (The pen pattern and mode are set up—as `gray` and `notPatXor`—to conform to Appearance-compliant human interface guidelines.)

A rectangle (in global coordinates) whose upper-left corner is aligned with the port rectangle of the window's graphics port is passed into the `param` parameter of your window definition function. Your grow image should be sized appropriately for the specified rectangle. As the user drags the mouse, the Window Manager sends repeated `wGrow` messages, so that you can change your grow image to match the changing mouse location.

`DrawGrowIcon` (page 13) draws a dotted (gray) outline of the window and also the lines delimiting the title bar, size box, and scroll bar areas.

Your window definition function should return 0 as the function result for this message.

Drawing the Size Box

When the Window Manager passes `wDrawGIcon` in the `message` parameter, your window definition function should respond by drawing the size box in the content region if the window is active. If the window is inactive, your window definition function should draw whatever is appropriate to show that the window cannot currently be sized. Your window definition function may also draw scroll bar delimiter lines. Your window definition function should ignore the `param` parameter.

If the size box is located in the window frame, draw the size box in response to a `wDraw` message, not a `wDrawGIcon` message.

Your window definition function should return 0 as the function result for this message.

Reporting Window Features

When the Window Manager passes `kWindowMsgGetFeatures` in the `message` parameter, your window definition function should respond by setting the `param` parameter to reflect the features that your window supports. The value passed back in the `param` parameter should be comprised of one or more of the following values:

```
enum{
    kWindowCanGrow              = (1 << 0),
    kWindowCanZoom              = (1 << 1),
    kWindowCanCollapse          = (1 << 2),
    kWindowIsModal              = (1 << 3),
    kWindowCanGetWindowRegion   = (1 << 4),
    kWindowIsAlert              = (1 << 5),
    kWindowHasTitleBar          = (1 << 6),
};
```

Constant descriptions

`kWindowCanGrow` If this bit (bit 0) is set, the window has a grow box (may not be visible).

`kWindowCanZoom` If this bit (bit 1) is set, the window has a zoom box (may not be visible).

`kWindowCanCollapse` If this bit (bit 2) is set, the window has a collapse box.

`kWindowIsModal` If this bit (bit 3) is set, the window should behave as modal.

`kWindowCanGetWindowRegion` If this bit (bit 4) is set, the window supports a call to `GetWindowRegion` (page 12).

`kWindowIsAlert` If this bit (bit 5) is set, the window is an alert box (may be movable or not). When this constant is added to `kWindowIsModal`, the user should be able to switch out of the application and move the alert box.

`kWindowHasTitleBar` If this bit (bit 6) is set, the window has a title bar.

Your window definition function should return 1 as the function result for this message.

Returning the Location of Window Regions

When the Window Manager passes `kWindowMsgGetRegion` in the `message` parameter, your window definition function should respond by returning the location (in global coordinates) of the specified window region. A pointer to a window region structure will be passed in the `param` parameter.

The window region structure is a structure of type `GetWindowRegionRec`.

```
struct GetWindowRegionRec {
    RgnHandle           winRgn;
    WindowRegionCode    regionCode;
```

```
};
typedef struct GetWindowRegionRec GetWindowRegionRec;
typedef GetWindowRegionRec *GetWindowRegionPtr;
```

Field descriptions

`winRgn`	A handle to a window region based on the value specified in the `regionCode` field. Modify this region.
`regionCode`	A value representing a given window region; see "Window Region Constants" (page 44).

Your window definition function should return an operating system status (`OSStatus`) message as the function result for this message. The result code `errWindowRgnInvalid` indicates that the window region passed in was not valid.

Window Manager Data Types

The following Window Manager data types are new, changed, or not recommended with Mac OS 8 or Appearance Manager 1.0:

- `WindowRecord` (page 28)
- `WinCTab` (page 29)
- `AuxWinRec` (page 29)
- `'WIND'` (page 29)
- `'wctb'` (page 31)
- `'WDEF'` (page 31)

WindowRecord

In Mac OS 8, you should use the color window structure instead of the window structure, since Color QuickDraw is always available with Mac OS 8. The `WindowRecord` structure is not recommended with Mac OS 8 and later.

WinCTab

The `WinCTab` (window color table) structure is not recommended with Appearance Manager 1.0 and later. When the Appearance Manager is available and you are using standard windows, all information in the window color table structure is ignored except the part identifier constant `wContentColor` in the `value` field of the `ColorSpec` structure. This constant produces the background color for the window's content region. If your are creating your own custom window definition functions, the window color table structure can still be used.

AuxWinRec

The `AuxWinRec` (auxiliary window) structure is not recommended with Appearance Manager 1.0 and later. When the Appearance Manager is available and you are using standard windows, most of the fields of the auxiliary window structure are ignored. In the future, standard system windows may not have entries in the auxiliary window list. If you are creating your own window definition function, the auxiliary window structure can still be used.

'WIND'

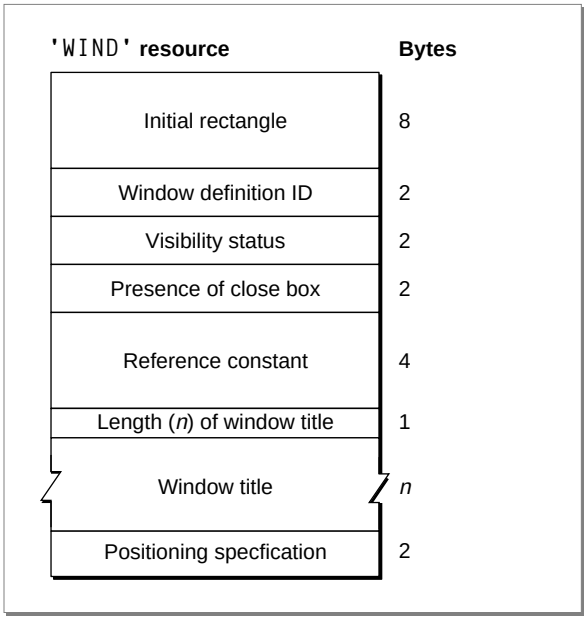
The window resource is changed with Appearance Manager 1.0 to support the Window Manager's new, standard window types. See "Window Definition IDs" (page 32) for a description of the Appearance-compliant window types.

You typically define a window (`'WIND'`) resource for each type of window that your application creates. Use `GetNewWindow` (page 10) or `GetNewCWindow` to create a window based on a `'WIND'` resource. Both functions create a new window structure and fill it in according to the values specified in the `'WIND'` resource.

Note
Window resources must have resource ID numbers greater than 127.

Figure 1-1 illustrates a compiled `'WIND'` resource.

Figure 1-1 Structure of a compiled window `('WIND')` resource



A compiled version of a window resource contains the following elements:

- The upper-left and lower-right corners, in global coordinates, of a rectangle that defines the initial size and placement of the window's content region. Your application can change this rectangle before displaying the window, either programmatically or through an optional positioning code described later in this section.
- The window definition ID. The window definition ID is an integer that contains the resource ID of the window definition function in its upper 12 bits and a variation code in its lower 4 bits; see "Window Definition IDs" (page 32) for a discussion of standard pre-Appearance and Appearance-compliant window definition IDs.
- A specification that determines whether the window will be visible or invisible. This characteristic controls only whether the Window Manager displays the window, not necessarily whether the window can be seen on the screen. (A visible window entirely covered by other windows, for example, is "visible" even though the user cannot see it.) You typically create a new

window in an invisible state, build the content area of the window, and then display the completed window.

- A specification that determines whether or not the window will have a close box. The `'WDEF'` draws the close box when it draws the window frame. The window type specified in the second field determines whether a window can support a close box; this field determines whether the close box is present.
- A reference constant, which your application can use for whatever data it needs to store. When it builds a new window structure, the Window Manager stores, in the `refCon` field, whatever value you specify in the fifth element of the window resource. You can also put a place-holder value here and then set the `refCon` field programmatically with the `SetWRefCon` function.
- A pascal string that specifies the window title.
- A positioning specification that overrides the window position established by the rectangle in the first field. The existence of this field is optional. The positioning constants are convenient when the user is creating new documents or when you are handling your own dialog boxes and alert boxes. When you are creating a new window to display a previously saved document, however, you should display the new window in the same rectangle as that in which it was previously displayed.

'wctb'

The window color table `('wctb')` resource is not recommended with Appearance Manager 1.0 and later. When the Appearance Manager is available and you are using standard windows, all information in the window color table resource is ignored except the part identifier constant `wContentColor` in the `value` field of the `ColorSpec` structure. This constant produces the background color for the window's content region.

If your are creating your own custom window definition functions, the window color table resource can still be used.

'WDEF'

The window definition (`'WDEF'`) resource is changed with Appearance Manager 1.0 to support the window definition message constants `kWindowMsgGetFeatures` and `kWindowMsgGetRegion`.

The window definition resource is the executable code for your window definition function. Provide as the resource data the compiled or assembled code of your window definition procedure. The entry point of your procedure must be at the beginning of the resource data.

See `MyWindowDefProc` (page 18) for more information about creating a window definition function.

▲ **WARNING**
All resources of type `'WDEF'` should be nonpurgeable, to ensure that the resource is always loaded. If the resource is not available when the Window Manager needs to load it, a crash will occur.

Window Manager Constants

The following Window Manager constants are new, changed, or not recommended with Mac OS 8 or Appearance Manager 1.0:

- "Window Definition IDs" (page 32)
- "Window Resource IDs" (page 41)
- "Window Definition Variation Codes" (page 43)
- "Window Region Constants" (page 44)
- "Part Identifier Constants" (page 45)
- "FindWindow Result Code Constants" (page 46)

Window Definition IDs

Window definition IDs are changed with Appearance Manager 1.0. The Window Manager now provides many new, standard, Appearance-compliant window types.

Your application typically supplies a window definition ID to a resource of type `'WIND'` (page 29) or to a window-creation function to specify which window definition function to use in creating the window. A variation code may also be used to describe variations of the same basic window.

The window definition ID is an integer that contains the resource ID of the window definition function in its upper 12 bits and a variation code in its lower 4 bits. For a given resource ID and variation code, the window definition ID is derived as follows:

window definition ID = (16 x resource ID) + variation code.

See “Window Resource IDs” (page 41) and “Window Definition Variation Codes” (page 43) for descriptions of possible values.

If you wish to create a custom window, you can write your own window definition function. For an example, see `MyWindowDefProc` (page 18).

IMPORTANT

The window definition IDs for dialog boxes and utility (floating) windows pertain to the appearances of these windows only, not their behaviors. For example, if you want a utility window to have the proper behavior, that is, float, your application must provide for it.

When mapping is enabled, standard pre-Appearance window definition function IDs will be mapped to their Appearance-compliant equivalents. Table 1-1 (page 34) lists the standard pre-Appearance and Appearance-compliant window definition ID constants.

Table 1-1 Pre-Appearance and Appearance-compliant window definition IDs

Pre-Appearance window	Appearance-compliant window	Description
dBoxProc	kWindowModalDialogProc	Modal dialog box
None	kWindowAlertProc	Modal alert box
movableDBoxProc	kWindowMovableModalDialogProc	Movable modal dialog box
None	kWindowMovableModalGrowProc	Movable modal dialog box with size box
None	kWindowMovableAlertProc	Movable modal alert box
plainDBox	kWindowPlainDialogProc	Modeless dialog box
altDBoxProc	kWindowShadowDialogProc	Modeless dialog box with shadow
noGrowDocProc	kWindowDocumentProc	Movable window with no size box
documentProc	kWindowGrowDocumentProc	Movable window with size box
zoomNoGrow	kWindowFullZoomDocumentProc	Movable window with full zoom box and no size box
zoomDocProc	kWindowFullZoomGrowDocumentProc	Movable window with full zoom box and size box
None	kWindowVertZoomDocumentProc	Movable window with vertical zoom box and no size box
None	kWindowVertZoomGrowDocumentProc	Movable window with vertical zoom box and size box
None	kWindowHorizZoomDocumentProc	Movable window with horizontal zoom box and no size box

Pre-Appearance window	Appearance-compliant window	Description
None	kWindowHorizZoomGrowDocumentProc	Movable window with horizontal zoom box and size box
rDocProc	None (use rDocProc)	Round-cornered window
floatProc	kWindowFloatProc	Utility window with no size box or zoom box
floatGrowProc	kWindowFloatGrowProc	Utility window with size box
floatZoomProc	kWindowFloatFullZoomProc	Utility window with full zoom box
floatZoomGrowProc	kWindowFloatFullZoomGrowProc	Utility window with full zoom box and size box
None	kWindowFloatVertZoomProc	Utility window with vertical zoom box
None	kWindowFloatVertZoomGrowProc	Utility window with vertical zoom box and size box
None	kWindowFloatHorizZoomProc	Utility window with horizontal zoom box
None	kWindowFloatHorizZoomGrowProc	Utility window with horizontal zoom box and size box
floatSideProc	kWindowFloatSideProc	Utility window with side title bar and no size box or zoom box
floatSideGrowProc	kWindowFloatSideGrowProc	Utility window with side title bar and size box

Pre-Appearance window	Appearance-compliant window	Description
floatSideZoomProc	kWindowFloatSideFullZoomProc	Utility window with side title bar and full zoom box
floatSideZoomGrowProc	kWindowFloatSideFullZoomGrowProc	Utility window with side title bar, size box, and full zoom box
None	kWindowFloatSideVertZoomProc	Utility window with side title bar and vertical zoom box
None	kWindowFloatSideVertZoomGrowProc	Utility window with side title bar, vertical zoom box, and size box
None	kWindowFloatSideHorizZoomProc	Utility window with side title bar and horizontal zoom box
None	kWindowFloatSideHorizZoomGrowProc	Utility window with side title bar, horizontal zoom box, and size box

```
enum {
    documentProc                        = 0,
    dBoxProc                            = 1,
    plainDBox                           = 2,
    altDBoxProc                         = 3,
    noGrowDocProc                       = 4,
    movableDBoxProc                     = 5,
    zoomDocProc                         = 8,
    zoomNoGrow                          = 12,
    rDocProc                            = 16,
    kWindowDocumentProc                 = 1024,
    kWindowGrowDocumentProc             = 1025,
    kWindowVertZoomDocumentProc         = 1026,
    kWindowVertZoomGrowDocumentProc     = 1027,
    kWindowHorizZoomDocumentProc        = 1028,
```

```
    kWindowHorizZoomGrowDocumentProc    = 1029,
    kWindowFullZoomDocumentProc         = 1030,
    kWindowFullZoomGrowDocumentProc     = 1031,
    kWindowPlainDialogProc              = 1040,
    kWindowShadowDialogProc             = 1041,
    kWindowModalDialogProc              = 1042,
    kWindowMovableModalDialogProc       = 1043,
    kWindowAlertProc                    = 1044,
    kWindowMovableAlertProc             = 1045,
    kWindowMovableModalGrowProc         = 1046,
    kWindowFloatProc                    = 1057,
    kWindowFloatGrowProc                = 1059,
    kWindowFloatVertZoomProc            = 1061,
    kWindowFloatVertZoomGrowProc        = 1063,
    kWindowFloatHorizZoomProc           = 1065,
    kWindowFloatHorizZoomGrowProc       = 1067,
    kWindowFloatFullZoomProc            = 1069,
    kWindowFloatFullZoomGrowProc        = 1071,
    kWindowFloatSideProc                = 1073,
    kWindowFloatSideGrowProc            = 1075,
    kWindowFloatSideVertZoomProc        = 1077,
    kWindowFloatSideVertZoomGrowProc    = 1079,
    kWindowFloatSideHorizZoomProc       = 1081,
    kWindowFloatSideHorizZoomGrowProc   = 1083,
    kWindowFloatSideFullZoomProc        = 1085,
    kWindowFloatSideFullZoomGrowProc    = 1087,
    floatProc                           = 1985,
    floatGrowProc                       = 1987,
    floatZoomProc                       = 1989,
    floatZoomGrowProc                   = 1991,
    floatSideProc                       = 1993,
    floatSideGrowProc                   = 1995,
    floatSideZoomProc                   = 1997,
    floatSideZoomGrowProc               = 1999
};
```

Constant descriptions

`documentProc`	Pre-Appearance document window (movable window with size box).
`dBoxProc`	Pre-Appearance modal dialog box.

`plainDBox` Pre-Appearance modeless dialog box.

`altDBoxProc` Pre-Appearance modeless dialog box with shadow.

`noGrowDocProc` Pre-Appearance movable window with no size box or zoom box.

`movableDBoxProc` Pre-Appearance movable modal dialog box.

`zoomDocProc` Pre-Appearance movable window with size box and full zoom box.

`zoomNoGrow` Pre-Appearance window with full zoom box and no size box.

`rDocProc` Pre-Appearance rounded-corner window. You can control the diameter of curvature of a rounded-corner window (window type `rDocProc`) by adding one of these integers to the `rDocProc` constant:

Window definition ID	Diameters of curvature
`rDocProc`	16, 16
`rDocProc + 2`	4, 4
`rDocProc + 4`	6, 6
`rDocProc + 6`	10, 10

`kWindowDocumentProc`
Appearance-compliant movable window with no size box or zoom box. Available with Appearance 1.0 and later.

`kWindowGrowDocumentProc`
Appearance-compliant standard document window (movable window with size box). Available with Appearance 1.0 and later.

`kWindowVertZoomDocumentProc`
Appearance-compliant window with vertical zoom box and no size box. Available with Appearance 1.0 and later.

`kWindowVertZoomGrowDocumentProc`
Appearance-compliant window with vertical zoom box and size box. Available with Appearance 1.0 and later.

`kWindowHorizZoomDocumentProc`
Appearance-compliant window with horizontal zoom box and no size box. Available with Appearance 1.0 and later.

`kWindowHorizZoomGrowDocumentProc`
: Appearance-compliant window with horizontal zoom box and size box. Available with Appearance 1.0 and later.

`kWindowFullZoomDocumentProc`
: Appearance-compliant window with full zoom box and no size box. Available with Appearance 1.0 and later.

`kWindowFullZoomGrowDocumentProc`
: Appearance-compliant window with full zoom box and size box. Available with Appearance 1.0 and later.

`kWindowPlainDialogProc`
: Appearance-compliant modeless dialog box. Available with Appearance 1.0 and later.

`kWindowShadowDialogProc`
: Appearance-compliant modeless dialog box with shadow. Available with Appearance 1.0 and later.

`kWindowModalDialogProc`
: Appearance-compliant modal dialog box. Available with Appearance 1.0 and later.

`kWindowMovableModalDialogProc`
: Appearance-compliant movable modal dialog box. Available with Appearance 1.0 and later.

`kWindowAlertProc`
: Appearance-compliant alert box. Available with Appearance 1.0 and later.

`kWindowMovableAlertProc`
: Appearance-compliant movable alert box. Available with Appearance 1.0 and later.

`kWindowMovableModalGrowProc`
: Appearance-compliant movable modal dialog box with size box. Available with Appearance 1.0.1 and later.

`kWindowFloatProc`
: Appearance-compliant utility (floating) window with no size box or zoom box. Available with Appearance 1.0 and later.

`kWindowFloatGrowProc`
: Appearance-compliant utility (floating) window with a size box. Available with Appearance 1.0 and later.

`kWindowFloatVertZoomProc`

Appearance-compliant utility (floating) window with a vertical zoom box. Available with Appearance 1.0 and later.

`kWindowFloatVertZoomGrowProc`

Appearance-compliant utility (floating) window with a vertical zoom box and size box. Available with Appearance 1.0 and later.

`kWindowFloatHorizZoomProc`

Appearance-compliant utility (floating) window with a horizontal zoom box. Available with Appearance 1.0 and later.

`kWindowFloatHorizZoomGrowProc`

Appearance-compliant utility (floating) window with a horizontal zoom box and size box. Available with Appearance 1.0 and later.

`kWindowFloatFullZoomProc`

Appearance-compliant utility (floating) window with full zoom box. Available with Appearance 1.0 and later.

`kWindowFloatFullZoomGrowProc`

Appearance-compliant utility (floating) window with full zoom box and size box. Available with Appearance 1.0 and later.

`kWindowFloatSideProc`

Appearance-compliant utility (floating) window with side title bar. Available with Appearance 1.0 and later.

`kWindowFloatSideGrowProc`

Appearance-compliant utility (floating) window with side title bar and size box. Available with Appearance 1.0 and later.

`kWindowFloatSideVertZoomProc`

Appearance-compliant utility (floating) window with side title bar and vertical zoom box. Available with Appearance 1.0 and later.

`kWindowFloatSideVertZoomGrowProc`

Appearance-compliant utility (floating) window with side title bar, vertical zoom box, and size box. Available with Appearance 1.0 and later.

`kWindowFloatSideHorizZoomProc`
: Appearance-compliant utility (floating) window with side title bar and horizontal zoom box. Available with Appearance 1.0 and later.

`kWindowFloatSideHorizZoomGrowProc`
: Appearance-compliant utility (floating) window with side title bar, horizontal zoom box, and size box. Available with Appearance 1.0 and later.

`kWindowFloatSideFullZoomProc`
: Appearance-compliant utility (floating) window with side title bar and full zoom box. Available with Appearance 1.0 and later.

`kWindowFloatSideFullZoomGrowProc`
: Appearance-compliant utility (floating) window with side title bar, full zoom box, and size box. Available with Appearance 1.0 and later.

`floatProc`
: Pre-Appearance utility (floating) window with no size box or zoom box.

`floatGrowProc`
: Pre-Appearance utility (floating) window with size box.

`floatZoomProc`
: Pre-Appearance utility (floating) window with zoom box.

`floatZoomGrowProc`
: Pre-Appearance utility (floating) window with size box and zoom box.

`floatSideProc`
: Pre-Appearance utility (floating) window with side title bar and no size or zoom box.

`floatSideGrowProc`
: Pre-Appearance utility (floating) window with side title bar and size box.

`floatSideZoomProc`
: Pre-Appearance utility (floating) window with side title bar and zoom box.

`floatSideZoomGrowProc`
: Pre-Appearance utility (floating) window with side title bar, size box, and zoom box.

Window Resource IDs

Window resource IDs are changed with Appearance Manager 1.0. The Window Manager now provides many new standard, Appearance-compliant window resource IDs for your program.

You can use a window resource ID constant to create a window definition ID; see “Window Definition IDs” (page 32) for more details.

Note that the standard Appearance-compliant resource ID constants `kWindowDocumentDefProcResID`, `kWindowUtilityDefProcResID`, and `kWindowUtilitySideTitleDefProcResID` specify windows with collapse boxes.

Note
Resource IDs 0 through 127 are reserved for use by the system.

```
enum {
    kStandardWindowDefinition           = 0,
    kRoundWindowDefinition              = 1,
    kWindowDocumentDefProcResID         = 64,
    kWindowDialogDefProcResID           = 65,
    kWindowUtilityDefProcResID          = 66,
    kWindowUtilitySideTitleDefProcResID = 67,
    kFloatingWindowDefinition           = 124
};
```

Constant descriptions

`kStandardWindowDefinition`
Defines pre-Appearance standard document windows and dialog boxes. When mapping is enabled, this resource ID is mapped to `kWindowDocumentDefProcResID` or `kWindowDialogDefProcResID`. When mapped to `kWindowDocumentDefProcResID`, this produces an Appearance-compliant standard document window with no size box and no vertical or horizontal zoom box. When mapped to `kWindowDialogDefProcResID`, this produces an Appearance-compliant dialog box with no size box and a 3-pixel space between the dialog box’s content and structure region.

`kRoundWindowDefinition`
Defines pre-Appearance standard desk-accessory style windows. This resource ID is not mapped to any Appearance-compliant resource ID when mapping is enabled.

`kWindowDocumentDefProcResID`
Defines Appearance-compliant standard document

windows with a size box. Standard document windows created with this resource ID can use variation codes to create windows with vertical and horizontal zoom boxes. Available with Appearance 1.0 and later.

`kWindowDialogDefProcResID`

Defines Appearance-compliant dialog and alert boxes. Modal and movable modal dialog boxes created with this resource ID are displayed with no space between their content and structure region. Alert boxes created with this resource ID are displayed with a red-tinged border. Available with Appearance 1.0 and later.

`kWindowUtilityDefProcResID`

Defines Appearance-compliant utility (floating) windows with a top title bar and a size box. Available with Appearance 1.0 and later.

`kWindowUtilitySideTitleDefProcResID`

Defines Appearance-compliant utility (floating) windows with a side title bar and a size box. Available with Appearance 1.0 and later.

`kFloatingWindowDefinition`

Defines pre-Appearance utility (floating) windows. When mapping is enabled, this resource ID is mapped to `kWindowUtilityDefProcResID` or `kWindowUtilitySideTitleDefProcResID`. When mapped to `kWindowUtilityDefProcResID`, this produces an Appearance-compliant utility window with no size box until `DrawGrowIcon` (page 13) is called. When mapped to `kWindowUtilitySideTitleDefProcResID`, it produces an Appearance-compliant utility window with a side title bar and no size box until `DrawGrowIcon` (page 13) is called.

Window Definition Variation Codes

When the Appearance Manager is available, you should use the Appearance-compliant window definition ID constants described in “Window Definition IDs” (page 32), rather than variation codes.

Window Region Constants

You can pass constants of type `WindowRegionCode` in the `inRegionCode` parameter of `GetWindowRegion` (page 12) to obtain a handle to a specific window region. The `WindowRegionCode` constants are available with Appearance Manager 1.0 and later.

Figure 1-2 (page 45) illustrates the location of these regions in a window.

```
enum {
    kWindowTitleBarRgn      = 0,
    kWindowTitleTextRgn     = 1,
    kWindowCloseBoxRgn      = 2,
    kWindowZoomBoxRgn       = 3,
    kWindowDragRgn          = 5,
    kWindowGrowRgn          = 6,
    kWindowCollapseBoxRgn   = 7,
    kWindowStructureRgn     = 32,
    kWindowContentRgn       = 33
};
typedef UInt16 WindowRegionCode;
```

Constant descriptions

`kWindowTitleBarRgn` The entire area occupied by a window's title bar, including the title text region.

`kWindowTitleTextRgn`
That portion of a window's title bar that is occupied by the name of the window.

`kWindowCloseBoxRgn` The area occupied by a window's close box.

`kWindowZoomBoxRgn` The area occupied by a window's zoom box.

`kWindowDragRgn` The draggable area of the window frame; this area includes the title bar and window outline and excludes the size box, close box, zoom box, and collapse box.

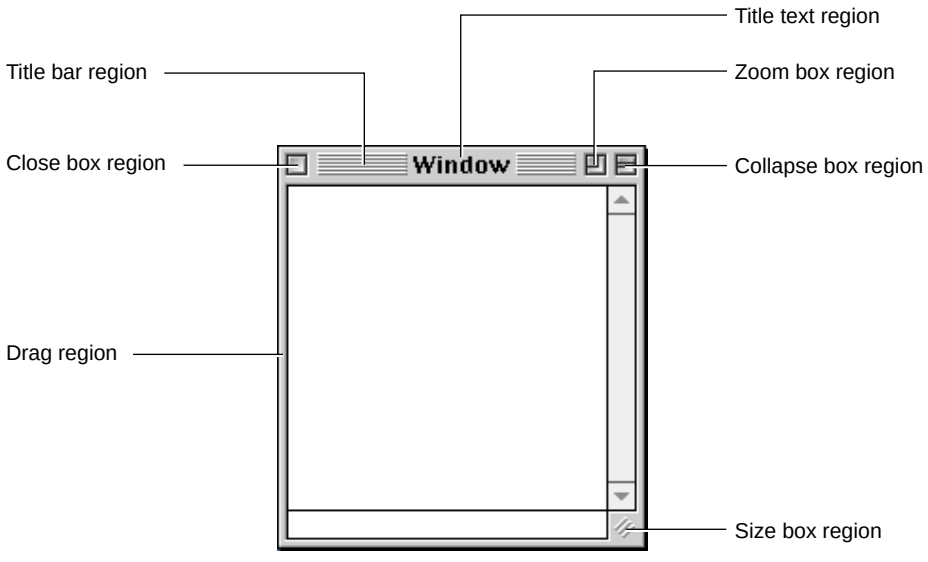
`kWindowGrowRgn` The area occupied by a window's size box.

`kWindowCollapseBoxRgn`
The area occupied by a window's collapse box.

`kWindowStructureRgn`
The entire area occupied by a window, including the frame and content region; the window may be partially off-screen but its structure region does not change.

`kWindowContentRgn` The window's content region—the part of a window in which your application displays the contents of the window or dialog, including the size box and any controls.

Figure 1-2 Window regions



Part Identifier Constants

When the Appearance Manager is available and you are using standard windows, all the fields of the window color table structure are ignored except the part identifier constant `wContentColor` in the `value` field of the `ColorSpec` structure, which produces the background color for the window's content region.

If you are creating your own custom windows, the window color table structure and all its part identifier constants can still be used.

FindWindow Result Code Constants

When your application receives a mouse-down event, you typically call the function `FindWindow` (page 11). `FindWindow` returns an integer that specifies the location, in global coordinates, of the cursor at the time the user pressed the mouse button. The `FindWindow` result code constants are changed with Appearance Manager 1.0 to include the `inCollapseBox` constant.

```
enum {
    inDesk          = 0,
    inMenuBar       = 1,
    inSysWindow     = 2,
    inContent       = 3,
    inDrag          = 4,
    inGrow          = 5,
    inGoAway        = 6,
    inZoomIn        = 7,
    inZoomOut       = 8,
    inCollapseBox   = 11
};
```

Constant descriptions

`inDesk` The cursor is in the desktop region, not in the menu bar, a driver window, or any window that belongs to your application. When `FindWindow` returns `inDesk`, your application doesn't need to do anything.

`inMenuBar` The user has pressed the mouse button while the cursor is in the menu bar. When `FindWindow` returns `inMenuBar`, your application typically adjusts its menus and then calls the Menu Manager function `MenuSelect` to let the user choose menu items.

`inSysWindow` The user has pressed the mouse button while the cursor is in a window belonging to a driver in your application's partition. If `FindWindow` returns `inSysWindow`, your application typically calls the function `SystemClick`.

`inContent` The user has pressed the mouse button while the cursor is in the content area (excluding the size box in an active window) of one of your application's windows. When `FindWindow` returns `inContent`, your application determines how to handle clicks in the content region.

`inDrag`	The user has pressed the mouse button while the cursor is in the drag region of a window. When `FindWindow` returns `inDrag`, your application typically calls `DragWindow` to let the user drag the window to a new location.
`inGrow`	The user has pressed the mouse button while the cursor is in an active window's size box. When `FindWindow` returns `inGrow`, your application typically calls `GrowWindow` .
`inGoAway`	The user has pressed the mouse button while the cursor is in an active window's close box. When `FindWindow` returns `inGoAway`, your application typically calls `TrackGoAway` to track mouse activity while the button is down and then calls its own function for closing a window if the user releases the button while the cursor is in the close box.
`inZoomIn`	The user has pressed the mouse button while the cursor is in the zoom box of an active window that is currently in the standard state. When `FindWindow` returns `inZoomIn`, your application typically calls `TrackBox` to track mouse activity while the button is down and then calls its own function for zooming a window if the user releases the button while the cursor is in the zoom box.
`inZoomOut`	The user has pressed the mouse button while the cursor is in the zoom box of an active window that is currently in the user state. When `FindWindow` returns `inZoomOut`, your application typically calls the function `TrackBox` to track mouse activity while the button is down. Your application then calls its own function for zooming a window if the user releases the button while the cursor is in the zoom box.
`inCollapseBox`	The user has pressed the mouse button while the cursor is in an active window's collapse box. When `FindWindow` returns `inCollapseBox`, your application typically does nothing, because the system will collapse your window for you. Available with Appearance Manager 1.0 and later.

Result Codes

The most common result codes that can be returned by Window Manager functions are listed below.

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough memory
`resNotFound`	–192	Unable to read resource

APPENDIX A

Version History

This document has had the following releases:

Table A-1 *Mac OS 8 Window Manager Reference* Revision History

Version	Notes
Nov. 11, 1998	Removed "Window Manager Reference" chapter from the *Mac OS 8 Toolbox Reference* document. *Mac OS 8 Window Manager Reference* is now available as an independent document. The following corrections were made: `IsWindowCollapsable` (page 17). Added description for previously undocumented function. "Window Definition IDs" (page 32). Added description for previously undocumented constant `kWindowMovableModalGrowProc`.
Jan. 15, 1998	The following corrections were made: Noted Appearance 1.0.2 where applicable. Window state data structure. Recategorized from "not recommended with the Appearance Manager" to "unchanged" and, therefore, removed from this delta document.
Dec. 10, 1997	The following corrections were made: "Window Region Constants" (page 44). Clarified the constant descriptions.
Dec. 2, 1997	PDF formatting improved.
Nov. 3, 1997	First document release.

APPENDIX

Version History

Index

M

N

P

R

S

U

W

Z

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITERS
Lisa Karpinski, Donna S. Lee, and Judith Rosado

ILLUSTRATORS
David Arrigoni and Karin Stroud

PRODUCTION EDITOR
Glen Frank

PROJECT MANAGER
Tony Francis

Acknowledgments to Matt Ackeret, Pete Gontier, Chris Thomas, and Ed Voas.