Libraries

COLLABORATORS

	TITLE : Libraries	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Libraries

1.1 Amiga® RKM Libraries: 4 Intuition Windows

This chapter provides a general description of windows: how to open windows and define their characteristics; how to get the system gadgets for shaping, moving, closing, and depth arranging windows; how to handle window I/O; and how to preserve the display when windows get overlapped.

1.2 4 Intuition Windows / About Windows

Windows are rectangular display areas that open on screens. The window acts as a virtual terminal allowing a program to interact with the user as if it had the entire display all to itself.

Each window opens on a specific screen and takes certain characteristics, such as resolution, colors and display attributes, from that screen. These values cannot be adjusted on a window by window basis. Other window characteristics such as the text font are inherited from the screen but can be changed.

An application may open several windows at the same time on a single screen. The Workbench and other public (shareable) screens allow windows opened by different applications to coexist on the same screen.

Windows are moveable and can be positioned anywhere within the screen on which they exist. Windows may also have a title and borders containing various gadgets for controlling the window.

1.3 4 / About Windows / Window System Gadgets

Each window may have a number of system gadgets which allow the user to control window size, shape and arrangement. These gadgets are: the drag bar, the depth gadget, the sizing gadget, the zoom gadget and the close gadget.

The drag bar allows the user to change the position of the window with respect to the screen. The drag bar is in the top border of a window and occupies any space in the top border that is not used by other gadgets. The window may be dragged left, right, up and down on the screen, with the limitation that the entire window must remain within the screen's boundaries. This is done by positioning the pointer over the title bar, selecting the window and dragging to the new position. Window drag may be cancelled by pressing the right mouse button before the drag is completed.

Figure 4-1: A Window with System Gadgets

The depth gadget allows the user to depth arrange a window with respect to other windows on the screen. The depth gadget is always positioned in the upper right corner of the window. Clicking the depth gadget will move the frontmost window behind all other windows. If the window is not the frontmost, it will be moved to the front. Selecting the depth gadget with the Shift qualifier always moves the window to the back (behind other windows).

The sizing gadget allows the user to change the size of the window. Sizing is subject to minimum and maximum values set by the application. Width and height are independent in a sizing operation. The sizing gadget is always positioned in the lower right corner of the window. It allows the user to drag this corner of the window to a new position relative to the upper left corner of the window, thus changing the width and height of the window. Window sizing using the sizing gadget may be cancelled by pressing the right mouse button before the size is completed.

The zoom gadget allows the user to quickly alternate between two preset window size and position values. The zoom gadget is always placed immediately to the left of the depth gadget. If there is no depth gadget on the window, the zoom gadget will still appear next to where the depth gadget would have been.

The close gadget performs no direct action on the window, rather it causes Intuition to send a message to the application to close the window. This allows the application to perform any required processing or to warn the user before it closes the window. The close gadget is always positioned in the upper left corner of the window.

1.4 4 / About Windows / The Active Window

There is only one window in the system active at any time. The active window receives all user input, including keyboard and mouse events. This is also known as the input focus, as all input is focused at this single point.

Some areas of the active window are displayed more boldly than those on inactive windows. The active window's borders are filled in with a color which is designed to stand out from the background while inactive windows have their borders filled with the background color. The specific coloring of active and inactive windows is dependent on the screen on which the window is opened. See the section "DrawInfo and the 3D Look" in the "Intuition Screens" chapter for more information.

Windows have two optional titles: one for the window and one for the screen. The window title appears in the top border of the window, regardless of whether the window is active or inactive. The window's screen title appears in the screen's title bar only when the window is active. This gives the user a secondary clue as to what application is active in the screen.

The active window's menus are displayed on the screen when the right mouse button (the menu button) is pressed. If the active window has no menus, then none will be displayed.

Each window may also have its own mouse-pointer image. Changing the active window will change the pointer to the one currently set for the new active window.

1.5 4 Intuition Windows / Basic Window Structures and Functions

This section introduces the basic data structures and functions an application uses to create an Intuition window. Intuition uses the Window data structure defined in <intuition/intuition.h> to represent windows. Most of Intuition's window functions use this structure in some way. Other related structures used to create and operate windows are summarized in Table 4-1.

Table 4-1: Data Structures Used with Intuition Windows

Structure Name	Description	Defined in Include File
Window	Main Intuition structure that defines a window	<intuition/intuition.h>
TagItem	General purpose parameter structure used to set up windows in V37	<utility/tagitem.h>
NewWindow	Parameter structure used to create a window in V34	<intuition/intuition.h>
ExtNewWindow	An extension to the NewWindow structure used in V37 for backward compatibility with older systems	<intuition/intuition.h>
Layer	A drawing rectangle that clips graphic operations	<graphics/clip.h>

```
                falling within its boundaries

  RastPort      General purpose handle used     <graphics/rastport.h>
                for graphics library drawing
                operations.
```

Intuition's window system relies on the layers library and graphics library to implement many of its features. The Window structure is closely related to the Layer structure defined in <graphics/clip.h> and the RastPort structure defined in <graphics/rastport.h>. The system uses these structures to store drawing state data. In general, applications don't have to worry about the internal details of these structures but use them instead as convenient handles, passing them as arguments to lower-level functions. See the "Layers Library" and "Graphics Primitives" chapters for more information.

```
 Opening a Window                Window Dimensions
 Closing Windows                 Window Border Dimensions
 Windows and Screens             Changing Window Size Limits
 Graphics and Text in Windows
```

1.6 4 / Basic Window Structures and Functions / Opening a Window

A window is opened and displayed by a call to one of the OpenWindow() functions: OpenWindow(), OpenWindowTagList() or OpenWindowTags().

```
    struct Window *OpenWindowTagList( struct NewWindow *newWindow,
                                      struct TagItem *tagList );
    struct Window *OpenWindowTags( struct NewWindow *newWindow,
                                   unsigned long tag1Type, ... );
    struct Window *OpenWindow( struct NewWindow *newWindow );
```

The type of window and its attributes are specified in NewWindow or TagItem structures depending on which function is used. These functions all return a pointer to a new Window structure if they succeed. A NULL return indicates failure.

OpenWindowTagList() and OpenWindowTags() are available only in Release 2 (V36) and later versions of the OS. For these functions, window attributes are specified in TagItem structures which are paired data items specifying an attribute and its setting. (See the 'Utility Library' chapter for more information on TagItems.)

OpenWindow() is available in all versions of the OS. Window attributes can be specified using a NewWindow structure but only a limited set of window attributes are available this way. To support both the new window features of Release 2 and compatibility with older versions of the OS, use OpenWindow() with an extended version of the NewWindow structure named ExtNewWindow. See the WFLG_NW_EXTENDED flag description in the "Window Attributes" section below for more information on using OpenWindow() with the extended NewWindow structure.

Further references to OpenWindow() in this chapter will apply to all three functions. These calls are the only proper method for allocating a Window

structure. The tag based versions are recommended for V36 and later versions of the OS. Use the ExtNewWindow structure with OpenWindow() to provide backward compatibility.

OpenWindowTagList() Example Setting Window Attributes

1.7 4 / / Opening A Window / Setting Window Attributes

Depending on which function is used to open a window, the window's attributes may be specified using TagItems, or a NewWindow structure or an ExtNewWindow structure. In the code above, the window attributes are set up with an array of TagItems:

```
struct TagItem win_tags[] =
    {
    {WA_Left,       MY_WIN_LEFT},
    {WA_Top,        MY_WIN_TOP},
    {WA_Width,      MY_WIN_WIDTH},
    {WA_Height,     MY_WIN_HEIGHT},
    {WA_CloseGadget,TRUE},
    {WA_IDCMP,      IDCMP_CLOSEWINDOW},
    {TAG_DONE, NULL},
    };
```

These window attributes set the window's position (WA_Left, WA_Top) and size (WA_Width, WA_Height), request a close gadget on the window (WA_CloseGadget) and ask Intuition to send a message whenever the user activates the close gadget (WA_IDCMP).

Throughout this chapter window attributes are referred to by their TagItem ID name (the name is always prefixed with "WA_"). See the section below on "Window Attributes" for a complete list.

```
Old and New Flag Names.
-----------------------
The names used for IDCMP flags and window flags have been changed
under Release 2.  IDCMP flag names are now preceded by "IDCMP_".
Likewise window flag names are now preceded by "WFLG_".  The old
names (and their new equivalents) are listed in
<intuition/iobsolete.h>.  You may want to refer to this file if
you are working with example code written for V34 and older versions
of the OS.
```

1.8 4 / Basic Window Structures and Functions / Closing Windows

Call the CloseWindow() function to close a window, remove its imagery from the display, and clean up any system resources used by the window. Typically, you call CloseWindow() when Intuition informs you that the user has selected the window's close gadget but this is not a requirement nor does the window have to be active to be closed.

```
void CloseWindow( struct Window *window );
```

Pass this function a pointer to the Window structure returned by one of the OpenWindow() calls.

If you call CloseWindow() on the active window, the previously active window (if available) will become the active window. If the previously active window has already closed, then the window active prior to that window will become the active window. (Applications should not rely on this behavior. To make a specific window become active, call the ActivateWindow() function.)

Intuition does not automatically close a window when the user selects the close window gadget. Instead, Intuition sends your program a message about the user's action. The program can then perform whatever cleanup is necessary before closing the window with the CloseWindow() function.

1.9 4 / Basic Window Structures and Functions / Windows and Screens

Windows may be opened on one of three screen types: a custom screen, a public screen or the Workbench screen. A custom screen is one created and controlled by your application. Once you have set up a custom screen, you may open a window on it directly by calling one of the three open window functions.

To open a window on a custom screen, call OpenWindowTagList() (or OpenWindowTags()) with the WA_CustomScreen tag along with a pointer to the custom screen. This must be a pointer to a screen created by your application. For systems prior to Release 2, use the OpenWindow() call with NewWindow.Type set to CUSTOMSCREEN and NewWindow.Screen set to a pointer to your custom screen.

You may choose to open a window on an existing public (shareable) screen instead of setting up your own custom screen. Such windows are often referred to as visitor windows because they "visit" a screen managed by the system or another application.

For Workbench or other public screens that are not created and managed directly by your application, you must lock the screen before opening the window. This ensures that the screen remains open while your call to open the window is processed. One way to obtain a lock on a public screen is by calling the LockPubScreen() function (see the "Intuition Screens" chapter).

Use WA_PubScreenName with NULL to open a visitor window on the default public screen (normally the Workbench screen). If a name is provided and the named screen exists, the visitor window will open on that named screen. In this case the system locks the named screen for you so there is no need to call LockPubScreen() directly. The open window call will fail if it cannot obtain a lock on the screen. If the WA_PubScreenFallBack tag is TRUE, the window will open on the default public screen when WA_PubScreenName can't be found.

Another method to open a visitor window on a public screen is to use the WA_PubScreen tag along with a pointer to the Screen structure of the public screen obtained via LockPubScreen().

The application may also request the name of the "next" public screen, which allows windows to "jump" between public screens. This is done by closing the application window on the first screen and opening a new window on the next screen. (See the "Intuition Screens" chapter for more information on public and custom screens.)

If no action is taken by the programmer to open the window on a specific screen, the window will open on the default public screen (normally the Workbench). This behavior is shown in the above example using OpenWindowTagList().

There are two global modes which come into play when a visitor window is opened on a public screen. If the global mode SHANGHAI is set, Workbench application windows will be opened on the default public screen. A second global mode, POPPUBSCREEN, forces a public screen to be moved to the front when a visitor window opens on it. These modes can be changed using SetPubScreenModes(), however, these should only be set according to the preferences of the user.

Simple Window on a Public Screen Example

1.10 4 / Window Structures and Functions / Graphics and Text in Windows

Applications can call functions in both the graphics library and the Intuition library to render images, lines, text and other graphic elements in windows. The graphics library provides primitive operations such as area fill, line drawing, text and animation.

The number of colors and the palette available in a window are defined by the screen in which the window opens. Applications should never change the palette of a screen unless the screen is a custom screen created by the application.

Graphics rendered into the window should respect the drawing pens defined for the screen. See the section on "DrawInfo and the 3D Look" in the "Intuition Screens" chapter for more information.

Default window fonts come from one of two places, depending on the screen on which the window opens. The window title font is always taken from the screen font. If the screen is opened with a font specified, either by specifying the tag SA_Font or the variable NewScreen.Font, then Window.RPort->Font is taken from the screen's font. Otherwise, the window's rastport's font is taken from GfxBase->DefaultFont. This information is available to the application if it opened the screen.

If the application did not open the screen, it has no way of knowing which font has been used for the window. Applications that require to know the window's font before the window is open must explicitly set the font (using SetFont()) for that window after opening it. In this case, the application may use any font it desires. It is recommended that applications use the screen's font if they support proportional fonts, and GfxBase->DefaultFont otherwise, as these fonts are generally the user's preference.

Intuition also provides a minimal high level interface to some of the functions in the Graphics library. This includes calls to draw lines, text and images. See the chapter entitled "Intuition Images, Line Drawing and Text," for more information about using Intuition to render graphics.

1.11 4 / Basic Window Structures and Functions / Window Dimensions

The initial position and dimensions of the window are defined in the OpenWindowTagList() call. These values undergo error checking before the window is actually opened on the screen. If the dimensions are too big, the window will fail to open. (Or, you can use the WA_AutoAdjust tag if you want Intuition to move or size your window to fit.)

Maximum and minimum size values may also be defined, but are not required If the window does not have a sizing gadget. In setting these dimensions, bear in mind the horizontal and vertical resolutions of the screen in which the window will open.

The maximum dimensions of the window are unsigned values and may legally be set to the maximum by using the value 0xFFFF, better expressed as "~0". Using this value for the maximum dimensions allows the window to be sized to the full screen.

A Display Sized Window Example

1.12 4 / / Window Dimensions / A Display Sized Window Example

A full screen window is not always desirable. If the user is working on a large, scrolling screen, they may only want a window the size of the visible display. The following example calculates the visible area on a screen and opens a window in that area. The example assumes that the screen display clip is as large or larger than text overscan (OSCAN_TEXT) which is set by the user. The window is opened in the text overscan area, not within the actual display clip that is used for the screen. Use QueryOverscan() to find the standard overscan rectangles (display clips) for a screen. Use the graphics library call VideoControl() to find the true display clip of the screen (see the chapter on "Graphics Primitives" for more information on VideoControl()). The ViewPortExtra structure contains the display clip information.

About Screen Coordinates.

The screen's actual position may not exactly equal the coordinates given in the LeftEdge and TopEdge fields of the Screen structure. This is due to hardware constraints that limit the fineness of the positioning of the underlying constructs. This may cause a window which is opened in the visible part of the screen to be incorrectly positioned by a small number of pixels in each direction. See the discussion of the screen's LeftEdge and TopEdge in the "Intuition Screens" chapter for more information.

```
     visiblewindow.c
```

1.13 4 / Basic Window Structures and Functions / Window Border Dimensions

Intuition automatically draws a border around a window unless directed otherwise, such as by setting the WFLG_BORDERLESS flag. Borderless windows may not have a window title or gadgets in the border (this includes the standard system gadgets). Otherwise they won't come out properly borderless.

The size of the border of an open window is available in the Window structure variables BorderLeft, BorderTop, BorderRight and BorderBottom. Intuition fills these in when the window is opened. To calculate the window border sizes before the window is opened you use information in the Screen structure as shown in the next listing.

```
    Gadgets Can Change Border Sizes.
    --------------------------------
    The following calculations do not take application border gadgets
    into account. If the program adds gadgets into the window's borders,
    Intuition will expand the borders to hold the gadgets.


if (NULL != (screen = LockPubScreen(NULL)))
    {
    top_border    = screen->WBorTop + screen->Font->ta_YSize + 1;
    left_border   = screen->WBorLeft;
    right_border  = screen->WBorRight;
    bottom_border = screen->WBorBottom;

    UnlockPubScreen(NULL,screen);
    }

/* if the sizing gadget is specified, then the border size must
** be adjusted for the border containing the gadget.  This may
** be the right border, the bottom border or both.
**
** We are using fixed values.  There is currently no system-approved
** method of finding this information before the window is opened.
** If you need to know these sizes BEFORE your window is opened,
** use the fixed values below.  Otherwise, use Window->BorderRight,
** etc. AFTER you have opened your window.
*/

/* values for non-lo-res screen */
right_border  = 18;  /* if sizing gadget in right border  */
bottom_border = 10;  /* if sizing gadget in bottom border */

/* values for lo-res screen */
right_border  = 13;  /* if sizing gadget in right border  */
bottom_border = 11;  /* if sizing gadget in bottom border */


Use the border sizes to position visual elements within the window.
Coordinates may be offset into the window by the size of the top and left
```

borders, for instance (x, y) becomes (x + BorderLeft, y + BorderTop). This may look clumsy, but it offers a way of avoiding a GimmeZeroZero window, which, although much more convenient to use, requires extra memory and degrades performance.

The right and bottom border values specify the width of these borders. The area within the borders of a window is defined as (BorderLeft, BorderTop) to (Width - 1 - BorderRight, Height - 1 - BorderBottom). The calculations subtract one from the height and width of the windows as positions count from zero, but dimensions count from one.

The window title bar is only available if one or more of the following is specified: window title, window drag gadget, window depth gadget, window close gadget or window zoom gadget. If none of these are specified, the top border will be much narrower.

Application gadgets may be added to the window border by setting a flag in the Gadget structure. A special flag must additionally be set to place gadgets into the borders of GimmeZeroZero windows. See the chapter "Intuition Gadgets," for more information about gadgets and their positioning. (Borderless windows have no visible border outlines and gadgets should not be placed in their borders.)

1.14 4 / Window Structures and Functions / Changing Window Size Limits

To change the sizing limits after the window has been opened, call WindowLimits() with the new values.

```
BOOL WindowLimits( struct Window *window, long widthMin, long heightMin,
                   unsigned long widthMax, unsigned long heightMax );
```

To maintain the current dimension, set the corresponding argument to 0. Out of range numbers are ignored. If the user is currently sizing the window, new limits take effect after the user releases the select button.

1.15 4 Intuition Windows / Communicating with Intuition

Intuition can notify an application when the user moves the mouse, makes a menu choice, selects an application gadget or changes the window's size. To find out about user activity from Intuition, there are two methods:

* Use the Intuition Direct Communications Message Port (IDCMP) system. Input events are received as standard Exec messages at a port Intuition creates for your window.

* Use the console.device to receive all input events as character sequences.

1.16 4 / Communicating with Intuition / The IDCMP

The IDCMP gives an application convenient access to many types of user input events through the Exec message and port system. Intuition input event messages include mouse and keyboard activity as well as high level events from menus and gadgets.

With the IDCMP, you specify the input events you want to know about when you open the window. The input events are specified with one or more of the IDCMP flags in <intuition/intuition.h>. Use the flags with the WA_IDCMP tag for the OpenWindowTagList() (or OpenWindowTags()) function. Or, set the flags in NewWindow.IDCMPFlags for the OpenWindow() function. If any IDCMP flags are set when the window is opened, Intuition automatically creates a message port for you to receive messages about user activity. If NULL is specified for IDCMP flags, no port is created. For more information on receiving messages from Intuition, see the IDCMP sections in the chapter "Intuition Input and Output Methods."

1.17 4 / Communicating with Intuition / The Console Device

An alternative to the message system used by the IDCMP is the console device. The console device gives your application input data translated to ASCII characters or ANSI escape sequences. Raw (untranslated) input is also available through the console device as ANSI escape sequences.

The console device also provides for convenient output of control codes and non-proportional (mono-spaced) text to the window. Output is character based, and includes capabilities such as automatic line wrapping and scrolling. The console device automatically formats and interprets the output stream. Output is kept within the window boundaries automatically so the application need not worry about overwriting the border (no GimmeZeroZero window required).

The console device must be opened by the application before it is used. See the chapter entitled "Intuition Input and Output Methods" or refer to the "Console Device" chapter of the Amiga ROM Kernel Reference Manual: Devices for more information about using the console device with your Intuition windows.

1.18 4 / Communicating with Intuition / The IDCMP And The Active Window

On the Amiga, all input is directed to a single window called the active window. In general, changing the active window should be left up to the user. (The user activates a window by pressing the select button while the pointer is within the window boundaries.) If the active window is changed, the user may be confused if the change was not performed at their direction. Hence, new windows should be activated only when they open as a direct and synchronous response to the user's action. Existing windows should almost never be activated by the application.

An application can learn when one of its windows is activated or deactivated by setting the IDCMP flags IDCMP_ACTIVEWINDOW and

IDCMP_INACTIVEWINDOW. When these flags are specified, the program will receive a message each time the user activates the window or causes the window to become inactive by activating some other window.

The application may specify that a window is to become active when it opens. This is done with the WA_Activate tag or by setting WFLG_ACTIVATE in NewWindow.Flags when the window is opened.

The application may also activate an existing window. This is done by calling the ActivateWindow() function, which will activate the window as soon as possible. Try to use this function only in response to user action since it may cause a shift in the input focus:

```
LONG ActivateWindow( struct Window *window );
```

This function call may have its action deferred. Do not assume that the selected window has become active when this call returns. Intuition will inform the application when this window has become active by sending an IDCMP_ACTIVEWINDOW message. Getting this message is the only supported way of tracking the activation status of your windows.

1.19 4 / Communicating with Intuition / The IDCMP And Gadgets

One way for a user to communicate with a program running under Intuition is through the use of gadgets. There are two basic kinds of gadgets: system gadgets, which are predefined and managed by Intuition, and application gadgets.

System Gadgets Application Gadgets

1.20 4 / / The IDCMP And Gadgets / System Gadgets

System gadgets on each window provide the user with the ability to manage the following aspects of the window: size, position and depth. These gadgets are managed by Intuition and the application does not need to take any action for them to operate properly. An additional system gadget is provided for the "close window" function. The close action is not directly managed by Intuition; selecting the close gadget will simply send a message to the application, which is responsible for closing the window.

All of these gadgets are optional, and independent of each other. The graphic representations of these gadgets are predefined, and Intuition always displays them in the same standard locations in the window borders.

The application may choose to be notified when the window changes size, or it may choose to control the timing of the sizing of the window. Controlling the timing of sizing operations is done through the use of the IDCMP_SIZEVERIFY message. IDCMP_SIZEVERIFY messages time out if the application does not respond fast enough. When these an IDCMP_SIZEVERIFY message times out the window sizing operation is cancelled by Intuition.

No information is available to the program on user changes to the depth

arrangement of a window. However a refresh message will be sent if part of the window needs to be redrawn as a result of a change to the depth arrangement.

Notification of changes to the position of the window or the size of the window are available through the IDCMP_CHANGEWINDOW and IDCMP_NEWSIZE flags. The application specifies the initial size, the maximum and minimum limits for sizing, and whether the sizing gadget is contained in the right border, bottom border or both borders. (See the section on "Border Dimensions" for information on how the specification of the sizing gadget affects the border sizes.)

The drag gadget has no imagery other than the implicit imagery of the title bar. Setting the window title does not interfere with drag gadget operation, nor does the drag gadget interfere with the display of the window title.

1.21 4 / / The IDCMP And Gadgets / Application Gadgets

The application may place gadgets in windows to request various kinds of input from the user. These gadgets may be specified in the OpenWindowTagList() call, or they may be created and added to the window later. For details about creating and using gadgets, see the chapters on "Intuition Gadgets" and the "GadTools Library".

1.22 4 Intuition Windows / Window Types

There are three special window types: Backdrop, Borderless and GimmeZeroZero. Backdrop windows stay anchored to the back of the display. Borderless windows have no borders rendered by Intuition. GimmeZeroZero windows provide clipping to protect the borders from graphics rendered into the window.

These window types can be combined, although the combinations are not always useful. For instance, a borderless, backdrop window can be created; however, a borderless, GimmeZeroZero window does not make sense. A window is not required to be any of these types.

Backdrop Window Type GimmeZeroZero Window Type
Borderless Window Type

1.23 4 / Window Types / Backdrop Window Type

Backdrop windows open behind all other non-backdrop windows, but in front of other backdrop windows that might already be open. Depth arrangement of a backdrop window affects the order of the window relative to other backdrop windows, but backdrop windows always stay behind all non-backdrop windows. No amount of depth arrangement will ever move a non-backdrop window behind a backdrop window.

The only system gadget that can be attached to a backdrop window is the closewindow gadget. Application gadgets are not restricted in backdrop windows.

Backdrop windows may often be used in place of drawing directly into the display memory of a custom screen. Such a technique is preferred, as backdrop windows are compatible with the Intuition windowing system. Using a backdrop window eliminates the danger of writing to the screen memory at a "bad" time or at the wrong position and overwriting data in a window.

To provide a full screen display area that is compatible with the windowing system, create a full sized, borderless, backdrop window with no system gadgets. Use the ShowTitle() call to hide or reveal the screen's title bar, as appropriate. See the Amiga ROM Kernel Reference Manual: Includes and Autodocs for a complete list of arguments for ShowTitle().

Backdrop windows are created by specifying the WFLG_BACKDROP flag or the WA_Backdrop tag in the OpenWindowTagList() call.

1.24 4 / Window Types / Borderless Window Type

The borderless window type has no borders rendered by Intuition. Such a window will have no visual delineation from the rest of the display. Be aware that a Borderless window which does not cover the entire display may cause visual confusion for the user. When using a borderless window that does not cover the entire display, the application should provide some form of graphics to replace the borders provided by Intuition.

In general, none of the system gadgets or the window title should be specified for a borderless window, as they may cause at least part of the border to be rendered.

A typical application of a borderless window is to simulate graphics drawn directly into the screen, while remaining compatible with windows and menus. In this case, the application will often create a full sized, borderless, backdrop window.

Use the WFLG_BORDERLESS flag or the WA_Borderless tag to get this window type.

1.25 4 / Window Types / GimmeZeroZero Window Type

GimmeZeroZero windows provide a window border layer separate from the main (inner) window layer. This allows the application to freely render into the window without worrying about the window border and its contents.

System gadgets and the window title are placed in the border layer. Application gadgets go into the inner window by default, but may be placed in the border. To position application gadgets in the border layer, the GTYP_GZZGADGET flag and the appropriate Gadget border flag must be set in the Activation field of the Gadget.

The top left coordinates of the inner window are always (0,0), regardless of the size or contents of the border, thus the name "GimmeZeroZero." The application need not take the border size into account when rendering. The inner window always begins at (0,0) and extends to (GZZWidth,GZZHeight). The GZZWidth and GZZHeight variables are available in the Window structure.

The GZZMouseX and GZZMouseY variables provide the position of the mouse relative to the inner window. Note that the mouse positions in IDCMP_MOUSEMOVE events are always relative to the total window, even for GimmeZeroZero windows.

Requesters in a GimmeZeroZero window are also positioned relative to the inner window. See the chapter entitled "Intuition Requesters and Alerts," for more information about requester location.

To specify a GimmeZeroZero window, set the WFLG_GIMMEZEROZERO flag or the WA_GimmeZeroZero tag in the OpenWindowTagList() call.

```
WARNING!
--------
The GimmeZeroZero window uses more system resources than other window
types because the window creates a separate layer for the border
display.  Using multiple GimmeZeroZero windows will quickly degrade
performance in the positioning and sizing of windows.

Applications should consider using regions as an alternative to
GimmeZeroZero windows.  See the "Layers Library" chapter,
especially the InstallClipRegion() function, for information on
setting up regions to limit graphics display in the window.
```

1.26 4 Intuition Windows / Preserving the Window Display

The layers library is what allows the display and manipulation of multiple overlapping rectangles, or layers. Intuition uses the layers library to manage its windows, by associating a layer to each window.

Each window is a virtual display. When rendering, the application does not have to worry about the current size or position of its window, and what other windows might be partly or fully obscuring its window. The window's RastPort is the handle to the its virtual display space. Intuition and graphics library rendering calls will recognize that this RastPort belongs to a layer, and act accordingly.

As windows are moved, resized, rearranged, opened, or closed, the on-screen representation changes. When part of a window which was visible now needs to appear in a new location, the layers library will move that imagery without involving the application. However, when part of a window that was previously obscured is revealed, or when a window is made larger, the imagery for the newly-visible part of the window needs to be redrawn. Intuition, through layers, offers three choices for how this is managed, trading off speed, memory usage, and application complexity.

* The most basic type of window is called Simple Refresh. When any

graphics operation takes place in this kind of window, the visible parts are updated, but rendering to the obscured parts is discarded. When the window arrangement changes to reveal a previously obscured part of such a window, the application must refresh that area.

* Alternately, a window may be made Smart Refresh, which means that when rendering occurs, the system will not only update the visible parts of the window, but it will maintain the obscured parts as well, by using off-screen buffers. This means that when an obscured part of the window is revealed, the system will restore the imagery that belongs there. The application needs only to refresh parts of the window that appear when the window is made bigger. Smart Refresh windows use more memory than Simple Refresh windows (for the storage of obscured areas), but they are faster.

* The third kind of window is called SuperBitMap. In such a window, the system can refresh the window even when it is sized bigger. For this to work, the application must store a complete bitmap for the window's maximum size. Such a window is more work to manage, and uses yet more memory. SuperBitMap windows are used less often than the other two types.

Intuition helps your application manage window refresh. First, Intuition will take care of redrawing the window border and any system and application gadgets in the window. Your application never has to worry about that. Second, Intuition will notify your application when it needs to refresh its window (by sending the IDCMP_REFRESHWINDOW event). Third, Intuition provides functions that restrict your rendering to the newly-revealed (damaged) areas only, which speeds up your refresh rendering and makes it look cleaner.

The Intuition, layers, and graphics libraries work together to make rendering into and managing windows easy. You obtain your windows through Intuition, which uses the Layers library to manage the overlapping, resizing, and re-positioning of the window layers. The layers library is responsible for identifying the areas of each window that are visible, obscured but preserved off-screen, or obscured and not preserved. The rendering functions in the graphics library and Intuition library know how to render into the multiple areas that layers library establishes.

Note that you may not directly manipulate layers on an Intuition screen. You cannot create your own layers on an Intuition screen, nor can you use the layers movement, sizing, or arrangement functions on Intuition windows. Use the corresponding Intuition calls instead. Some other Layers library calls (such as the locking calls) are sometimes used on Intuition screens and windows.

Damage Regions
Refreshing Intuition Windows
Intuition Refresh Events
Optimized Window Refreshing
Setting Up a SuperBitMap Window

1.27 4 / Preserving the Window Display / Damage Regions

The layers library and Intuition maintain a damage region for each window, which is the part of the window whose imagery is in need of repair, or

refreshing. Several things can add areas of the window to the damage region:

* Revealing an obscured part of a Simple Refresh window adds that area to the damage region

* Sizing a Simple or Smart Refresh window bigger along either axis adds the new area to the damage region

* Resizing a Simple or Smart Refresh window (smaller or bigger) adds the old and new border areas, and the areas occupied by certain gadgets (those whose position or size depend on window size) to the damage region.

1.28 4 / Preserving the Window Display / Refreshing Intuition Windows

When the user or an application performs an Intuition operation which causes damage to a window, Intuition notifies that window's application. It does this by sending a message of the class IDCMP_REFRESHWINDOW to that window's IDCMP.

In response to this message, your application should update the damaged areas. Rendering proceeds faster and looks cleaner if it is restricted to the damaged areas only. The BeginRefresh()/EndRefresh() pair achieve that. The application should call BeginRefresh() for the window, and then do its rendering. Any rendering that would have gone into undamaged areas of the window is automatically discarded; only the area in need of repair is affected. Finally, the application should call EndRefresh(), which removes the restriction on rendering, and informs the system that the damage region has been dealt with. Even if your application intends to do no rendering, it must at least call BeginRefresh()/EndRefresh(), to inform the system that the damage region is no longer needed. If your application never needs to render in response to a refresh event, it can avoid having to call BeginRefresh()/EndRefresh() by setting the WFLG_NOCAREREFRESH flag or the WA_NoCareRefresh tag in the OpenWindowTagList() call.

Note that by the time that your application receives notification that refresh is needed, Intuition will have already refreshed your window's border and all gadgets in the window, as needed. Thus, it is unnecessary to use any of the gadget-refreshing functions in response to an IDCMP_REFRESHWINDOW event.

Operations performed between the BeginRefresh()/EndRefresh() pair should be restricted to simple rendering. All of the rendering functions in Intuition library and Graphics library are safe. Avoid RefreshGList() or RefreshGadgets(), or you risk deadlocking the computer. Avoid calls that may lock the LayerInfo or get complicated in Intuition, since BeginRefresh() leaves the window's layer or layers locked. Avoid AutoRequest() and EasyRequest(), and therefore all direct or indirect disk related DOS calls. See the "Intuition Gadgets" chapter for more information on gadget restrictions with BeginRefresh()/EndRefresh().

Simple Refresh Smart Refresh SuperBitMap Refresh

1.29 4 / / Refreshing Intuition Windows / Simple Refresh

For a Simple Refresh window, only those pixels actually on-screen are maintained by the system. When part of a Simple Refresh window is obscured, the imagery that was there is lost. As well, any rendering into obscured portions of such a window is discarded.

When part of the window is newly revealed (either because the window was just made larger, or because that part used to be obscured by another window), the application must refresh any rendering it wishes to appear into that part. The application will learn that refresh is needed because Intuition sends an IDCMP_REFRESHWINDOW event.

1.30 4 / / Refreshing Intuition Windows / Smart Refresh

If a window is of the Smart Refresh type, then the system will not only preserve those pixels which are actually on-screen, but it will save all obscured pixels that are within the current window's size. The system will refresh those parts of the window revealed by changes in the overlapping with other windows on the screen, without involving the application. However, any part of the window revealed through the sizing of the window must be redrawn by the application. Again, Intuition will notify the application through the IDCMP_REFRESHWINDOW event.

Because the obscured areas are kept in off-screen buffers, Smart Refresh windows are refreshed faster than Simple Refresh windows are, and often without involving the application. Of course, for the same reason, they use more display memory.

1.31 4 / / Refreshing Intuition Windows / SuperBitMap Refresh

The SuperBitMap refresh type allows the application to provide and maintain bitmap memory for graphics in the window. The bitmap can be any size as long as the window sizing limits respect the maximum size of the bitmap.

SuperBitMap windows have their own memory for maintaining all obscured parts of the window up to the size of the defined bitmap, including those parts outside of the current window. Intuition will update all parts of the window that are revealed through changes in sizing and changes in window overlapping. The application never needs to redraw portions of the window that were revealed by sizing or positioning windows in the screen.

SuperBitMap windows require the application to allocate a bitmap for use as off-screen memory, instead of using Intuition managed buffers. This bitmap must be as large as, or larger than, the inner window's maximum dimensions (that is, the window's outside dimensions less the border sizes).

SuperBitMap windows are almost always WFLG_GIMMEZEROZERO, which renders the borders and system gadgets in a separate bitmap. If the application wishes to create a SuperBitMap window that is not GimmeZeroZero, it must

make the window borderless with no system gadgets, so that no border imagery is rendered by Intuition into the application's bitmap.

1.32 4 / Preserving the Window Display / Intuition Refresh Events

When using a Simple Refresh or a Smart Refresh windows, the program may receive refresh events, informing it to update the display. See the above discussion for information on when refresh events are sent.

A message of the class IDCMP_REFRESHWINDOW arrives at the IDCMP, informing the program of the need to update the display. The program must take some action when it receives a refresh event, even if it is just the acceptable minimum action described below.

On receiving a refresh event, BeginRefresh() must be called, then the program should redraw its display, and, finally, call EndRefresh(). The minimum required action is to call the BeginRefresh()/EndRefresh() pair. This allows Intuition and the Layers library keep things sorted and organized.

1.33 4 / Preserving the Window Display / Optimized Window Refreshing

Bracketing the display updating in the BeginRefresh()/EndRefresh() pair automatically restricts all rendering to the "damaged" areas.

```
void BeginRefresh( struct Window *window );
void EndRefresh  ( struct Window *window, long complete );
```

These functions makes sure that refreshing is done in the most efficient way, only redrawing those portions of the window that really need to be redrawn. The rest of the rendering commands are discarded.

Operations performed between the BeginRefresh()/EndRefresh() pair should be restricted to simple rendering. All of the rendering functions in Intuition library and Graphics library are safe. Calls to RefreshGadgets() are not permitted. Avoid calls that may lock the LayerInfo, or get complicated in Intuition, since BeginRefresh() leaves the window's layer or layers locked. Avoid AutoRequest(), and therefore all direct or indirect disk related DOS calls. See the "Intuition Gadgets" chapter for more information on gadget restrictions with BeginRefresh()/EndRefresh().

Certain applications do not need to receive refresh events, and can avoid having to call BeginRefresh() and EndRefresh() by setting the WFLG_NOCAREREFRESH flag or the WA_NoCareRefresh tag in the OpenWindowTagList() call.

The EndRefresh() function takes a boolean value as an argument (complete in the prototype above). This value determines whether refreshing is completely finished. When set to FALSE, further refreshing may be performed between subsequent BeginRefresh()/ EndRefresh() pairs. Set the boolean to TRUE for the last call to EndRefresh().

It is critical that applications performing multiple BeginRefresh()/EndRefresh() pairs using EndRefresh(win,FALSE) hold layers locked through the entire process. The layer lock may only be released after the final call to EndRefresh(win,TRUE). See the "Layers Library" for more details.

The procedures outlined in this section take care of refreshing what is inside the window. Another function named RefreshWindowFrame() refreshes window borders, including the title region and gadgets:

```
void RefreshWindowFrame( struct Window *window );
```

Applications can use this function to update window borders after overwriting them with graphics.

1.34 4 / Preserving the Window Display / Setting Up A SuperBitMap Window

SuperBitMap windows are created by setting the WFLG_SUPER_BITMAP flag, or by specifying the WA_SuperBitMap tag in the OpenWindowTagList() call. A pointer to an allocated and initialized BitMap structure must be provided.

A SuperBitMap window requires the application to allocate and initialize its own bitmap. This entails allocating a BitMap structure, initializing the structure and allocating memory for the bit planes.

Allocate a BitMap structure with the Exec AllocMem() function. Then use the graphics function InitBitMap() to initialize the BitMap structure:

```
void InitBitMap( struct BitMap *bitMap, long depth,
                 long width, long height );
```

InitBitMap() fills in fields in the BitMap structure describing how a linear memory area is organized as a series of one or more rectangular bit-planes.

Once you have allocated and initialized the BitMap structure, use the graphics library function AllocRaster() to allocate the memory space for all the bit planes.

```
PLANEPTR AllocRaster( unsigned long width, unsigned long height );
```

The example listed in the next section shows how to allocate a BitMap structure, initialize it with InitBitMap() and use AllocRaster() function to set up memory for the bitplanes.

Graphics and Layers Functions for SuperBitMap Windows
SuperBitMap Window Example

1.35 4 / / Setting Up A SuperBitMap Window / Graphics and Layers Functions

The portion of the bitmap showing within a SuperBitMap window is controlled by the application. Initially, the window shows the bitmap starting from its origin (0,0) and clipped to fit within the window layer. The visible portion of the bitmap can be scrolled around within the window using the layers library ScrollLayer() function:

```
    void ScrollLayer(LONG unused, struct Layer *layer, LONG dx, LONG dy)
```

Pass this function a pointer to the window's layer in layer and the scroll offsets in dx and dy. (A pointer to the window's layer can be obtained from Window.RPort->Layer.)

When rendering operations are performed in a SuperBitMap window, any rendering that falls outside window boundaries is done in the application's bitmap. Rendering that falls within window bounds is done in the screen's bitmap. Before performing an operation such as a save on the application bitmap, the graphics library function SyncSBitMap() should be called:

```
    void SyncSBitMap(struct Layer *layer)
```

Pass this function a pointer to the window's layer. SyncSBitMap() copies the window contents to the corresponding part of the application bitmap, bringing it up to date. (If no rendering operations have been performed this call is not necessary.)

Similarly, after making any changes to the application bitmap such as loading a new one, the window's layer should be locked and the CopySBitMap() function should be called.

```
    void CopySBitMap(struct Layer *)
```

This function copies the new information in the appropriate area of the underlying bitmap to the window's layer.

For more information about bitmaps and layers, see the "Graphics Primitives" and "Layers Library" chapters of this manual. Also see the <graphics/clip.h>, <graphics/gfx.h>, <graphics/layers.h>, graphics library and layers library sections of the Amiga ROM Kernel Reference Manual: Includes and Autodocs.

1.36 4 Intuition Windows / The Window Structure

The Window structure is the main Intuition data structure used to represent a window. For the most part, applications treat this structure only as a handle. Window operations are performed by calling system functions that take Window as an argument instead of directly manipulating fields within the structure. However, there are some useful variables in a Window structure which are discussed in this section.

```
   struct Window
       {
       struct Window *NextWindow;
       WORD LeftEdge, TopEdge, Width, Height;
```

```
        WORD MouseY, MouseX;
        WORD MinWidth, MinHeight;
        UWORD MaxWidth, MaxHeight;
        ULONG Flags;
        struct Menu *MenuStrip;
        UBYTE *Title;
        struct Requester *FirstRequest, *DMRequest;
        WORD ReqCount;
        struct Screen *WScreen;
        struct RastPort *RPort;
        BYTE BorderLeft, BorderTop, BorderRight, BorderBottom;
        struct RastPort *BorderRPort;
        struct Gadget *FirstGadget;
        struct Window *Parent, *Descendant;
        UWORD *Pointer;
        BYTE PtrHeight, PtrWidth;
        BYTE XOffset, YOffset;
        ULONG IDCMPFlags;
        struct MsgPort *UserPort, *WindowPort;
        struct IntuiMessage *MessageKey;
        UBYTE DetailPen, BlockPen;
        struct Image *CheckMark;
        UBYTE *ScreenTitle;
        WORD GZZMouseX, GZZMouseY, GZZWidth, GZZHeight;
        UBYTE *ExtData;
        BYTE *UserData;
        struct Layer *WLayer;
        struct TextFont *IFont;
        ULONG MoreFlags;
        };
```

LeftEdge, TopEdge, Width and Height
    These variables reflect current position and size of the window. If the user sizes or positions the window, then these values will change. The position of the window is relative to the upper left corner of the screen.

MouseX, MouseY, GZZMouseX, GZZMouseY
    The current position of the Intuition pointer with respect to the window, whether or not this window is currently the active one. For GimmeZeroZero windows, the GZZ variables reflect the position relative to the inner layer (see "Window Types" below). For normal windows, the GZZ variables reflect the position relative to the window origin after taking the borders into account.

ReqCount
    Contains a count of the number of requesters currently displayed in the window. Do not rely on the value in this field, instead use IDCMP_REQSET and IDCMP_REQCLEAR to indirectly determine the number of open requesters in the window.

WScreen
    A pointer to the Screen structure of the screen on which this window was opened.

RPort
    A pointer to this window's RastPort structure. Use this RastPort

pointer to render into your window with Intuition or graphics library rendering functions.

BorderLeft, BorderTop, BorderRight, BorderBottom
These variables describe the actual size of the window borders. The border size is not changed after the window is opened.

BorderRPort
With GimmeZeroZero windows, this variable points to the RastPort for the outer layer, in which the border gadgets are kept.

UserData
This pointer is available for application use. The program can attach a data block to this window by setting this variable to point to the data.

For a commented listing of the Window structure see <intuition/intuition.h> in the Amiga ROM Kernel Reference Manual: Includes and Autodocs.

1.37 4 Intuition Windows / Window Attributes

This section discusses all window attributes. As mentioned earlier, a window's attributes may be specified with either TagItems, NewWindow or ExtNewWindow depending on how the window is opened.

Attributes are listed here by their TagItem ID name (TagItem.ti_Tag). For each tag item, the equivalent field setting in the NewWindow structure is also listed if it exists. Some window attributes specified with tags are available only in Release 2 and have no NewWindow equivalent.

1.38 4 / Window Attributes / Extended New Window

Of the three functions for opening a window, only OpenWindow() is present in all versions of the OS. This function takes a NewWindow structure as its sole argument. In order to allow applications to use the OpenWindow() call with Release 2 TagItem attributes, an extended version of the NewWindow structure has been created named ExtNewWindow.

Setting WFLG_NW_EXTENDED in the NewWindow.Flags field specifies to the OpenWindow() call that this NewWindow structure is really an ExtNewWindow structure. This is simply a standard NewWindow structure with a pointer to a tag list at the end. Since WFLG_NW_EXTENDED is ignored prior to V36, information provided in the tag list will be ignored by earlier versions of Intuition. Note that WFLG_NW_EXTENDED may not be specified in the WA_Flags tag.

1.39 4 / Window Attributes / Window Attribute Tags

WA_Left, WA_Top, WA_Width and WA_Height
Describe where the window will first appear on the screen and how large it will be initially. These dimensions are relative to the top left corner of the screen, which has the coordinates (0,0).

WA_Left is the initial x position, or offset, from the left edge of the screen. The leftmost pixel is pixel 0, and values increase to the right. Equivalent to NewWindow.LeftEdge.

WA_Top is the initial y position, or offset, from the top edge of the screen. The topmost pixel is pixel 0, and values increase to the bottom. Equivalent to NewWindow.TopEdge.

WA_Width is the initial window width in pixels. Equivalent to NewWindow.Width.

WA_Height is the initial window height in lines. Equivalent to NewWindow.Height.

WA_DetailPen and WA_BlockPen
WA_DetailPen specifies the pen number for the rendering of window details like gadgets or text in the title bar. WA_BlockPen specifies the pen number for window block fills, like the title bar. These pens are also used for rendering menus. Equivalent to NewWindow.DetailPen and NewWindow.BlockPen.

The specific color associated with each pen number depends on the screen. Specifying -1 for these values sets the window's detail and block pen the same as the screen's detail and block pen.

Detail pen and block pen have largely been replaced starting with V36 by the pen array in the DrawInfo structure. See the section on "DrawInfo and the 3D Look" in the "Intuition Screens" chapter for more information.

WA_IDCMP
IDCMP flags tell Intuition what user input events the application wants to be notified about. The IDCMP flags are listed and described in the OpenWindowTagList() description in the Amiga ROM Kernel Reference Manual: Includes and Autodocs and in the chapter "Intuition Input and Output Methods" in this book. Equivalent to NewWindow.IDCMPFlags.

If any of these flags are set, Intuition creates a pair of message ports for the window (one internal to Intuition and one used by the application). These ports are for handling messages about user input events. If WA_IDCMP is NULL or unspecified, no IDCMP is created for this window.

The ModifyIDCMP() function can be used to change the window's IDCMP flags after it is open.

WA_Gadgets
A pointer to the first in the linked list of Gadget structures that

are to be included in this window. These gadgets are application gadgets, not system gadgets. See the "Intuition Gadgets" chapter for more information. Equivalent to NewWindow.FirstGadget.

WA_Checkmark
A pointer to an Image structure, which is to be used as the checkmark image in this window's menus. To use the default checkmark, do not specify this tag or set this field to NULL. Equivalent to NewWindow.CheckMark.

WA_Title
A pointer to a NULL terminated text string, which is used as the window title and is displayed in the window's title bar.

Intuition draws the text using the colors defined in the DrawInfo pen array (DrawInfo.dri_Pens) and displays as much as possible of the window title, depending upon the current width of the title bar. Equivalent to NewWindow.Title. See the section on "DrawInfo and the 3D Look" in the "Intuition Screens" chapter for more information on the pen array.

The title is rendered in the screen's default font.

A title bar is added to the window if any of the properties WA_DragBar (WFLG_DRAGBAR), WA_DepthGadget (WFLG_DEPTHGADGET), WA_CloseGadget (WFLG_CLOSEGADGET) or WA_Zoom are specified, or if text is specified for a window title. If no text is provided for the title, but one or more of these system gadgets are specified, the title bar will be blank. Equivalent to NewWindow.Title.

WA_ScreenTitle
A pointer to a NULL terminated text string, which is used as the screen title and is displayed, when the window is active, in the screen's title bar. After the screen has been opened the screen's title may be changed by calling SetWindowTitles() (which is the only method of setting the window's screen title prior to V36).

WA_CustomScreen
A pointer to the Screen structure of a screen created by this application. The window will be opened on this screen. The custom screen must already be opened when the OpenWindowTagList() call is made. Equivalent to NewWindow.Screen, also implies NewWindow.Type of CUSTOMSCREEN.

WA_MinWidth, WA_MinHeight, WA_MaxWidth and WA_MaxHeight
These tags set the minimum and maximum values to which the user may size the window. If the flag WFLG_SIZEGADGET is not set, then these variables are ignored. Values are measured in pixels. Use (~0) for the WA_MaxWidth (WA_MaxHeight) to allow for a window as wide (tall) as the screen. This is the complete screen, not the visible part or display clip.

Setting any of these variables to 0, will take the setting for that dimension from its initial value. For example, setting MinWidth to 0, will make the minimum width of this window equal to the initial width of the window.

```
    Equivalent to NewWindow.MinWidth, NewWindow.MinHeight,
    NewWindow.MaxWidth and NewWindow.MaxHeight.  Use the WindowLimits()
    function to change window size limits after the window is opened.

WA_InnerWidth and WA_InnerHeight
    Specify the dimensions of the interior region of the window, i.e.,
    inside the border, independent of the border widths.  When using
    WA_InnerWidth and WA_InnerHeight an application will probably want to
    set WA_AutoAdjust.

WA_PubScreen
    Open the window as a visitor window on the public screen whose
    address is in the ti_Data field of the WA_PubScreen TagItem.  To
    ensure that this screen remains open until OpenWindowTagList() has
    completed, the application must either be the screen's owner, have a
    window open on the screen, or use LockPubScreen().  Setting this tag
    implies screen type of PUBLICSCREEN.

WA_PubScreenName
    Declares that the window is to be opened as a visitor on the public
    screen whose name is pointed to by the ti_Data field of the
    WA_PubScreenName TagItem.  The OpenWindowTagList() call will fail if
    it cannot obtain a lock on the named public screen and no fall back
    name (WA_PubScreenFallBack) is specified.  Setting this tag implies
    screen type of PUBLICSCREEN.

WA_PubScreenFallBack
    A Boolean, specifies whether a visitor window should "fall back" to
    the default public screen (or Workbench) if the named public screen
    isn't available This tag is only meaningful when used in conjunction
    with WA_PubScreenName.

WA_Zoom
    Pointer to an array of four WORDs, the initial LeftEdge, TopEdge,
    Width and Height values for the alternate zoom position and size.  It
    also specifies that the application wants a zoom gadget for the
    window, whether or not it has a sizing gadget.

    A zoom gadget is always supplied to a window if it has both depth and
    sizing gadgets.  This tag allows the application to open a window
    with a zoom gadget when the window does not have both the depth and
    sizing gadgets.

WA_MouseQueue
    An initial value for the mouse message backlog limit for this window.
    The SetMouseQueue() function will change this limit after the window
    is opened.

WA_RptQueue
    An initial value of repeat key backlog limit for this window.
```

1.40 4 / Window Attributes / Boolean Window Attribute Tags

```
These boolean window tags are alternatives to the NewWindow.Flags bit
fields with similar names.  Unlike the tags discussed above, the ti_Data
```

field of these TagItems is set to either TRUE or FALSE.

WA_SizeGadget
    Specifying this flag tells Intuition to add a sizing gadget to the window. Intuition places the sizing gadget in the lower right corner of the window. By default, the right border is adjusted to accommodate the sizing gadget, but the application can specify one of the following two flags to change this behavior. The WFLG_SIZEBRIGHT flag puts the sizing gadget in the right border. The WFLG_SIZEBBOTTOM flag puts the sizing gadget in the bottom border. Both flags may be specified, placing the gadget in both borders. Equivalent to NewWindow.Flags WFLG_SIZEGADGET.

WA_SizeBRight
    Place the size gadget in the right border. Equivalent to NewWindow.Flags WFLG_SIZEBRIGHT.

WA_SizeBBottom
    Place the size gadget in the bottom border. Equivalent to NewWindow.Flags WFLG_SIZEBBOTTOM.

WA_DragBar
    This flag turns the entire title bar of the window into a drag gadget, allowing the user to position the window by clicking in the title bar and dragging the mouse. Equivalent to NewWindow.Flags WFLG_DRAGBAR.

WA_DepthGadget
    Setting this flag adds a depth gadget to the window. This allows the user to change the window's depth arrangement with respect to other windows on the screen. Intuition places the depth gadget in the upper right corner of the window. Equivalent to NewWindow.Flags WFLG_DEPTHGADGET.

WA_CloseGadget
    Setting this flag attaches a close gadget to the window. When the user selects this gadget, Intuition transmits a message to the application. It is up to the application to close the window with a CloseWindow() call. Intuition places the close gadget in the upper left corner of the window. Equivalent to NewWindow.Flags WFLG_CLOSEGADGET.

WA_ReportMouse
    Send mouse movement events to the window as x,y coordinates. Also see the description of the IDCMP flag IDCMP_MOUSEMOVE, in the chapter "Intuition Input and Output Methods." Equivalent to NewWindow.Flags WFLG_REPORTMOUSE.

    The WFLG_REPORTMOUSE flag in the Flags field of the Window structure may be modified on the fly by the program. Changing this flag must be done as an atomic operation. Most compilers generate atomic code for operations such as window->flags |= WFLG_REPORTMOUSE or window->flags &= ~WFLG_REPORTMOUSE. If you are unsure of getting an atomic operation from your compiler, you may wish to do this operation in assembler, or bracket the code with a Forbid()/Permit() pair.

The use of the ReportMouse() function is strongly discouraged, due to historic confusion over the parameter ordering.

WA_NoCareRefresh
This window does not want IDCMP_REFRESHWINDOW events. Set this flag to prevent the window from receiving refresh window messages. Equivalent to NewWindow.Flags WFLG_NOCAREREFRESH. Intuition will manage BeginRefresh() and EndRefresh() internally.

WA_Borderless
Open a window with no borders rendered by Intuition. Equivalent to NewWindow.Flags WFLG_BORDERLESS.

Use caution setting this flag, as it may cause visual confusion on the screen. Also, some borders may be rendered if any of the system gadgets are requested, if text is supplied for the window's title bar, or if any of application gadgets are in the borders.

WA_Backdrop
Make this window a Backdrop window. Equivalent to NewWindow.Flags WFLG_BACKDROP.

WA_GimmeZeroZero
Set this tag to create a GimmeZeroZero window. GimmeZeroZero windows have the window border and border gadgets rendered into an extra layer. This extra layer slows down window operations, thus it is recommended that applications only use GimmeZeroZero windows when they are required. For clipping graphics to the area within the borders of a window, see the discussion of "Regions" in the "Layers Library" chapter. Equivalent to NewWindow.Flags WFLG_GIMMEZEROZERO.

WA_Activate
Activate the window when it opens. Equivalent to NewWindow.Flags WFLG_ACTIVATE. Use this flag carefully, as it can change where the user's input is going.

WA_RMBTrap
Catch right mouse button events for application use. Set this flag to disable menu operations for the window. When set, right mouse button events will be received as IDCMP_MOUSEBUTTONS with the MENUUP and MENUDOWN qualifiers. Equivalent to NewWindow.Flags WFLG_RMBTRAP.

The WFLG_RMBTRAP flag in the Window structure Flags field may be modified on the fly by the program. Changing this flag must be done as an atomic operation, as Intuition can preempt a multistep set or clear operation. An atomic operation can be done in assembler, using 68000 instructions that operate directly on memory. If you are unsure of generating such an instruction, place the operation within a Forbid()/Permit() pair. This will ensure proper operation by disabling multitasking while the flag is being changed.

WA_SimpleRefresh
The application program takes complete responsibility for updating the window. Only specify if TRUE. Equivalent to NewWindow.Flags WFLG_SIMPLE_REFRESH.

WA_SmartRefresh

Intuition handles all window updating, except for parts of the window revealed when the window is sized larger. Only specify if TRUE. Equivalent to NewWindow.Flags WFLG_SMART_REFRESH.

WA_SmartRefresh windows without a sizing gadget will never receive refresh events due to the user sizing the window. However, if the application sizes the window through a call like ChangeWindowBox(), ZipWindow() or SizeWindow(), a refresh event may be generated. Use WA_NoCareRefresh to disable refresh events.

WA_SuperBitMap
This is a pointer to a BitMap structure for a SuperBitMap window. The application will be allocating and maintaining its own bitmap. Equivalent to NewWindow.BitMap. Setting this tag implies the WFLG_SUPER_BITMAP property.

For complete information about SuperBitMap, see "Setting Up a SuperBitMap Window" in this chapter.

WA_AutoAdjust
Allow Intuition to change the window's position and dimensions in order to fit it on screen. The window's position is adjusted first, then the size. This property may be especially important when using WA_InnerWidth and WA_InnerHeight as border size depends on a user specified font.

WA_MenuHelp (new for V37, ignored by V36)
Enables IDCMP_MENUHELP: pressing Help during menus will return IDCMP_MENUHELP message. See the "Intuition Menus" chapter for more information.

WA_Flags
Multiple initialization of window flags, equivalent to NewWindow.Flags. Use the WFLG_ constants to initialize this field, multiple bits may be set by ORing the values together.

WA_BackFill
Allows you to specify a backfill hook for your window's layer. See the description of CreateUpFrontHookLayer() in the "Includes and Autodocs" manual. Note that this tag is implemented in V37, contrary to what some versions of the include files may say.

1.41 4 Intuition Windows / Other Window Functions

This section contains a brief overview of other Intuition functions that affect windows. For a complete description of all Intuition functions, see the Amiga ROM Kernel Reference Manual: Includes and Autodocs.

Custom Pointers

1.42 4 / Other Window Functions / Menus and the Active Window

Menus for the active window will be displayed when the user presses the menu button on the mouse. Menus may be disabled for the window by not providing a menu strip, or by clearing the menus with ClearMenuStrip(). Similarly, if the active window has WFLG_RMBTRAP set, the menu button will not bring up the menus.

Two other functions, SetMenuStrip() and ResetMenuStrip(), are used to attach or update the menu strip for a window.

```
void ClearMenuStrip( struct Window *window );
BOOL SetMenuStrip( struct Window *window, struct Menu *menu );
BOOL ResetMenuStrip( struct Window *window, struct Menu *menu );
```

If SetMenuStrip() has been called for a window, ClearMenuStrip() must be called before closing the window. After ClearMenuStrip() has been called, the user can no longer access menus for this window. See the chapter "Intuition Menus," for complete information about setting up menus.

1.43 4 / Other Window Functions / Requesters in the Window

Requesters are temporary sub-windows, usually containing several gadgets, used to confirm actions, access files, or adjust the options of a command the user has just given. Request() creates and activates a requester in the window. EndRequest() removes the requester from the window.

```
BOOL Request( struct Requester *requester, struct Window *window );
void EndRequest( struct Requester *requester, struct Window *window );
```

For simple requesters in a format that matches system requesters, two new functions have been added to Release 2:

```
LONG EasyRequestArgs( struct Window *window,
                      struct EasyStruct *easyStruct,
                      ULONG *idcmpPtr, APTR args );
LONG EasyRequest( struct Window *window,
                      struct EasyStruct *easyStruct,
                      ULONG *idcmpPtr, APTR arg1, ... );
```

The EasyRequest() functions support requesters with one or more gadgets automatically providing a layout that is sensitive to the current font and screen resolution. See the chapter "Intuition Requesters and Alerts" for more information on using requester functions.

1.44 4 / Other Window Functions / Program Control of Window Arrangement

MoveWindow(), SizeWindow(), WindowToFront() and WindowToBack() allow the program to modify the size and placement of its windows. These calls are available in all versions of the operating system.

MoveWindowInFrontOf(), ChangeWindowBox() and ZipWindow() have been added in Release 2 to provide more flexible control over the size and placement of windows.

All of these functions are asynchronous. The window will not be affected by them immediately, rather, Intuition will act on the request the next time it receives an input event. Currently this happens at a minimum rate of ten times per second, and a maximum of sixty times per second. There is no guarantee that the operation has taken place when the function returns. In some cases, there are IDCMP messages which will inform the application when the change has completed (for example, an IDCMP_NEWSIZE event indicates that a resize operation has completed).

Use the MoveWindow() function to move a window to a new position in the screen. Use SizeWindow() to change the size of the window:

```
    void MoveWindow( struct Window *window, long dx, long dy );
    void SizeWindow( struct Window *window, long dx, long dy );
```

Note that both MoveWindow() and SizeWindow() take the amount of change in each axis (delta values instead of absolute coordinates). To specify the coordinates as absolute numbers, use ChangeWindowBox(). The SizeWindow() function will respect the window's maximum and minimum dimensions only if the window has a sizing gadget.

A new function in Release 2, ChangeWindowBox(), allows an application to change the window size and position in a single call:

```
    void ChangeWindowBox( struct Window *window, long left, long top,
                          long width, long height );
```

Note that the position and size values are absolutes and not deltas. The window's maximum and minimum dimensions are always respected.

To depth arrange windows under program control, use WindowToFront() and WindowToBack():

```
    void WindowToFront( struct Window *window );
    void WindowToBack( struct Window *window );
```

WindowToFront() depth arranges a given window in front of all other windows on its screen. WindowToBack() depth arranges a given window behind all other windows on its screen.

To move a window in front of a specific, given window (as opposed to all windows), use MoveWindowInFrontOf():

```
    void MoveWindowInFrontOf( struct Window *window,
                              struct Window *behindWindow );
```

MoveWindowInFrontOf() is a new call provided in Release 2 and is not available in older versions of the OS.

To toggle the window size between its two zoom settings use ZipWindow(). This performs the same action that occurs when the user selects the zoom gadget:

```
void ZipWindow( struct Window *window );
```

The two zoom settings are the initial size and position of the window when it was first opened and the alternate position specified with the WA_Zoom tag. If no WA_Zoom tag is provided, the alternate position is taken from the window's minimum dimensions, unless the window was opened at its minimum dimension. In that case, the alternate position is taken from the window's maximum dimension. ZipWindow() is a new call provided in Release 2 and is not available in older versions of the OS.

1.45 4 / Other Window Functions / Changing the Window or Screen Title

Each window has its own window title and local screen title. The window title, if specified, is always displayed in the window. The local screen title, if specified, is only displayed in the screen's title bar when the window is active. If the window does not specify a local screen title, then the default screen title is used in the screen title bar when this window is active.

```
void SetWindowTitles( struct Window *window, UBYTE *windowTitle,
                      UBYTE *screenTitle );
```

This function changes the window title or local screen title for the given window. Both windowTitle and screenTitle can be set to -1, NULL or a NULL terminated string. Specifying -1 will not change the title from the current value. Specifying NULL will clear the window title or reset the screen title to the default title for the screen.

1.46 4 / Other Window Functions / Changing Message Queue Limits

Starting with V36, windows have limits on the number of mouse movement and repeat key messages that may be waiting at their IDCMP at any time. These queue limits prevent the accumulation of these messages, which may arrive at the IDCMP message port in large numbers.

Once a queue limit is reached, further messages of that type will be discarded by Intuition. The application will never hear about the discarded messages; they are gone forever. (Note that only mouse move and key repeat messages are limited this way. Other types of messages will still be added to the port.) Messages of the limited type will arrive at the port again after the application has replied to one of the messages in the queue.

The queue limits are independent of each other. Having reached the limit for one type of message does not prevent other types of messages (that have not yet reached their queuing limits) from being added to the IDCMP. Note that the queues apply only to the IDCMP and not to messages received

directly via an input handler or from the console device.

Order of event arrival is not a factor in the message count. Messages may be sequential or interspersed with other events--only the number of messages of the specific type waiting at the IDCMP matters.

The WA_RptQueue tag allows setting an initial value for the repeat key backlog limit for the window. There is no function to change this value as of V37. The default value for WA_RptQueue is 3.

The WA_MouseQueue tag allows setting an initial value for the mouse message backlog limit for the window. The default value for WA_MouseQueue is 5. The number may later be changed with a call to SetMouseQueue():

```
LONG SetMouseQueue( struct Window *window, unsigned long queueLength );
```

Note that real information may be lost if the queue fills and Intuition is forced to discard messages. See the chapter "Intuition Mouse and Keyboard" for more information.

1.47 4 / Other Window Functions / Changing Pointer Position Reports

Pointer position messages to a window may be turned on and off by simply setting or clearing the WFLG_REPORTMOUSE flag bit in Window->Flags, in an atomic way, as explained for the WA_RMBTrap tag in the "Window Attributes" section above. Using this direct method of setting the flag avoids the historic confusion on the ordering of the arguments of the ReportMouse() function call.

Mouse reporting may be turned on even if mouse movements were not activated when the window was opened. The proper IDCMP flags must be set for the window to receive the messages. See the chapter "Intuition Mouse and Keyboard" for more details on enabling mouse reporting in an application.

1.48 4 / Other Window Functions / Custom Pointers

The active window also has control over the pointer. If the active window changes the image for the pointer using the functions SetPointer() or ClearPointer(), the pointer image will change:

```
void SetPointer( struct Window *window, UWORD *pointer, long height,
                 long width, long xOffset, long yOffset );

void ClearPointer( struct Window *window );
```

SetPointer() sets up the window with a sprite definition for a custom pointer. If the window is active, the change takes place immediately. The pointer will not change if an inactive window calls SetPointer(). In this way, each window may have its own custom pointer that is displayed only when the window is active.

ClearPointer() clears the custom pointer from the window and restores it to the default Intuition pointer, which is set by the user. Setting a pointer for a window is discussed further in the chapter "Intuition Mouse and Keyboard".

1.49 4 Intuition Windows / Function Reference

The following are brief descriptions of the Intuition functions that relate to the use of Intuition windows. See the Amiga ROM Kernel Reference Manual: Includes and Autodocs for details on each function call.

Table 4-2: Functions for Intuition Windows

Function	Description
OpenWindowTagList()	Open a window.
OpenWindowTags()	Alternate calling sequence for OpenWindowTagList().
OpenWindow()	Pre-V36 way to open a window.
CloseWindow()	Close a window.
BeginRefresh()	Turn on optimized window refresh mode.
EndRefresh()	Turn off optimized window refresh mode.
RefreshWindowFrame()	Redraw the borders and border gadgets of an open window.
ActivateWindow()	Make an open window active.
SizeWindow()	Change the size of an open window.
MoveWindow()	Change the position of an open window.
ChangeWindowBox()	Change the size and position of an open window.
WindowLimits()	Change the minimum and maximum sizes of an open window.
WindowToBack()	Move a window behind all other windows.
WindowToFront()	Move a window in front of all other windows.
MoveWindowInFrontOf()	Move a window in front of another window.
ZipWindow()	Change the size of window to its alternate size.
SetWindowTitles()	Change the window titles for the window and the screen.
SetPointer()	Set up a custom pointer to display whenever the window is active.
ClearPointer()	Restore the mouse pointer to its default imagery.