Appendix A

INTUITION FUNCTION CALLS

In this appendix, all of the Intuition functions are presented in alphabetical order. The description of each function follows the format shown below:

NAME
The name of the function and a one-line description of what it does.

SYNOPSIS
The correct form of the function call.

FUNCTION SUMMARY

AddGadget	Adds a gadget to the gadget list of the window.
AllocRemember	Calls AllocMem() and creates a link node
AutoRequest	Automatically builds and gets response from a requester
BeginRefresh	Sets up a window for optimized refreshing
BuildSysRequest	Builds and displays a system requester
ClearDMRequest	Clears the DMRequest of the window
ClearMenuStrip	Clears the menu strip from the window
ClearPointer	Clears the pointer definition from a window
CloseScreen	Closes an Intuition screen
CloseWindow	Closes an Intuition window
CloseWorkBench	Closes the Workbench screen
CurrentTime	Gets the current time values
Display Alert	Creates a display of an alert message
DisplayBeep	"Beeps" the video display
DoubleClick	Tests two time values for double-click timing
DrawBorder	Draws the specified border into the RastPort
Drawlmage	Draws the specified image into the RastPort
EndRefresh	Ends the optimized refresh state of the window
EndRequest	Ends the request and resets the window
FreeRemember	Frees memory allocated by calls to AllocRemember()
FreeSysRequest	Frees up memory used by a call to BuildSysRequestQ
GetDefPrefs	Gets a copy of the the Intuition default Preferences
GetPrefs	Gets the current setting of the Intuition Preferences
InitRequester	Initializes a Requester structure
IntuiTextLength	Returns the length (pixel width) of an IntuiText
ItemAddress	Returns the address of the specified Menultem
MakeScreen	Does an Intuition-integrated MakeVPort() of a custom screen
ModifylDCMP	Modifies the state of the window's IDCMP
ModifyProp	Modifies the current parameters of a proportional gadget
MoveScreen	Attempts to move the screen by the delta amounts

MoveWindow	Asks Intuition to move a window
OffGadget	Disables the specified gadget
OffMenu	Disables the given menu or menu item
OnGadget	Enables the specified gadget
OnMenu	Enables the given menu or menu item
OpenScreen	Opens an Intuition screen
OpenWindow	Opens an Intuition window
OpenWorkBench	Opens the Workbench screen
PrintlText	Prints the text according to the IntuiText argument
RefreshGadgets	Refreshes (redraws) the gadget display
RemakeDisplay	Remakes the entire Intuition display
RemoveGadget	Removes a gadget from a window or a screen
ReportMouse	Tells Intuition whether or not to report mouse movement
Request	Activates a requester
RethinkDisplay	The grand manipulator of the entire Intuition display
ScreenToBack	Sends the specified screen to the back of the display
ScreenToFront	Brings the specified screen to the front of the display
SetDMRequest	Sets the DMRequest of the window
SetMenuStrip	Attaches the menu strip to the window
SetPointer	Sets a window with its own pointer
SetWindowTitles	Sets the window's titles for both window and screen
ShowTitle	Sets the screen title bar display mode
SizeWindow	Asks Intuition to size a window
ViewAddress	Returns the address of the Intuition View structure
ViewPortAddress	Returns the address of a window's ViewPort structure
WBenchToBack	Sends the Workbench Screen in back of all screens
WBenchToFront	Brings the Workbench Screen in front of all screens
WindowLimits	Sets the minimum and maximum limits of the window
WindowToBack	Asks Intuition to send this window to the back
WindowToFront	Asks Intuition to bring this window to the front

FUNCTION

Everything the function doss.

RESULT

Tk results, if an> returned by the function.

BUGS

Al known bugs, limitations, and deficiencies.

SEE ALSO

References to related functions in this and other books, as well as references to the teit of this book.

NAME

AddGadget — Adds a gadget to the gadget list of the window or screen.

SYNOPSIS

AddGadget(Pointer, Gadget, Position);
AO A l DO

FUNCTION

Adds the specified gadget to the gadget list of the given window, linked in at the position in the list specified by the **Position** argument (that is, if **Position** == **0,** the gadget will be inserted at the head of the list, and if **Position** == 1, the gadget will be inserted after the first gadget and before the second). If the **Position** you specify is greater than the number of gadgets in the list, your gadget will be added to the end of the list. This procedure returns the position at which your gadget was added.

Calling **AddGadgetQ** does not cause your gadget to be displayed. The benefit of this is that you may add several gadgets without having the gadget list redrawn every time. The drawback is that you are obliged to call **RefreshGadgets()** to have your added gadgets displayed.

NOTE: A relatively safe way to add the gadget to the end of the list is to specify a **Position** of -1. That way, only the 65,536th (and multiples of it) will be inserted at the wrong position. The return value of the procedure will tell you where it was actually inserted.

NOTE: The system window and screen gadgets are initially added to the front of the gadget list. The reason for this is: if you position your own gadgets in some way that interferes with the graphical representation of the system gadgets, the system's gadgets will be "hit" first by the user. If you then start adding gadgets to the front of the list, you will disturb this plan, so beware. On the other hand, if you do not violate the design rule of never overlapping your gadgets, there is no problem.

INPUTS

Pointer = pointer to the window to get your gadget.
Gadget = pointer to the new gadget.
Position = integer position in the list for the new gadget (starting from zero as the first position in the list).

RESULT

Returns the position where the gadget was actually added.

BUGS

None.

SEE ALSO

RemoveG adget ().

NAME

AllocRemember - Calls AllocMem() and creates a link* node.

SYNOPSIS

```
AllocRemember(RememberKey, Size, Flags);
              A0           D0    D1
```

FUNCTION

This routine calls the Exec **AllocMem()** function **for** you; it also links the parameters of the allocation into a master list, **so** that you can simply call the Intuition routine **FreeRemember()** at a later time to deallocate all allocated memory without being required to remember the details of the memory you have allocated.

This routine has two primary uses:

- Say that you are doing a long series of allocations in a procedure (such as the Intuition **OpenWindow()** procedure). If any one of the allocations fails for lack of memory, you need to abort the procedure. Abandoning ship correctly involves freeing up any memory you may have already allocated. This procedure allows you to free up that memory easily, without being required to keep track of how many allocations you have already done, what the sizes of the allocations were, or where the memory was allocated.
- Also, in the more general case, you may do all of the allocations in your entire program using this routine. Then, when your program is exiting, you can free it all up at once with a simple call to **FreeRemember().**

You create the "anchor" for the allocation master list by creating a variable that is a pointer to the **Remember** structure and initializing that pointer to NULL. This is called the **RememberKey.** Whenever you call **AllocRemember**(), the routine actually does two memory allocations, one for the memory you want and the other for a copy of a **Remember** structure. The **Remember** structure is filled in with data describing your memory allocation, and it is linked into the master list pointed to by your **RememberKey.** Then, to free up any memory that has been allocated, all you have to do is call **FreeRemember()** with your **RememberKey**.

Please read the **FreeRemember()** function description. As you will see, you can choose to free just the link nodes and keep all the allocated memory for yourself, or you can elect to free both the nodes and your memory buffers.

See the *Amiga ROM Kernel Manual* for a description of the AllocMem() call and the values you should use for the **Size** and **Flags** variables.

INPUTS

RememberKey = the address of a pointer to a Remember struct
the first call to AllocRememberQ, initialize this pointer to
instance:

```
struct Remember *RememberKey;
RememberKey = NULL;
AllocRemember(&RememberKey, BUFSIZE, MEMF_CHIP);
FreeRemember(&RememberKey, TRUE);
```

Size = the size in bytes of the memory allocation. Please refer
AllocMemQ function in the *Amiga ROM Kernel Manual* for
Flags = the specifications for the memory allocation. Please refer
AllocMemQ function in the *Amiga ROM Kernel Manual* for

RESULT

If the memory allocation is successful, this routine returns the b
your requested memory block. Also, the node to your block will
the list pointed to by your RememberKey variable. If the allocat
routine returns NULL and the list pointed to by RememberKey, i
undisturbed.

BUGS

None.

SEE ALSO

FreeRememberQ.
The Exec AllocMemQ function.

NAME

AutoRequest — Automatically builds and gets **response from a** requester.

SYNOPSIS

```
AutoRequest(Window, BodyText, PositiveText, NegativeText,
            A0      A1        A2            A3
      PositiveFlags, NegativeFlags, Width, Height);
            D0             D1        D2     D3
```

FUNCTION

This procedure automatically builds **a** requester **for** you and then waits for **a** response from the user or the system to satisfy your request. If the response is positive, this procedure returns **TRUE.** **If** the response is negative, this procedure returns FALSE.

This procedure first preserves the state **of the IDCMP** values of the window argument. Then it creates an **IDCMPFlag** specification by merging **your PositiveFlags, NegativeFlags,** and the IDCMP class GADGETUP. You may choose to specify no flags **for** either the **PositiveFlags** or **NegativeFlags** arguments.

The **IntuiText** arguments and the **Width and Height** values are passed directly to the **BuildSysRequest()** procedure, along with your window pointer and the IDCMP flags. Please **refer** to **BuildSysRequest() for** a description of the **IntuiText** that you are expected to supply when calling this routine. It is an important but long-winded description that need not be duplicated here.

If the **BuildSysRequest()** procedure does not return **a** pointer to a window, it will return TRUE or FALSE (not valid structure pointers) instead, and these BOOL values will be returned immediately.

On the other hand, if a valid window pointer is returned, that window will have had its IDCMP ports and flags initialized according to your specifications. **AutoRequest()** then waits for an IDCMP message on the **UserPort;** this message will satisfy one of three requirements:

- o If the message is of a class that matches one of your **PositiveFlags** arguments (if you have supplied any), this routine returns TRUE.
- o If the message class matches one of your **NegativeFlags** arguments (if you have supplied any), this routine returns FALSE.
- o The only other possibility is that the IDCMP message is of class GADGETUP, which means that one of the two gadgets, as specified by the **PositiveText** and **NegativeText** arguments, was selected by the user. If the TRUE gadget was selected, TRUE is returned. If the FALSE gadget was selected, FALSE is returned.

When the dust has settled, this routine calls FreeSysRequestQ, if necessary, to clean up the requester and any other allocated memory.

INPUTS

Window = pointer to a Window structure.
Body Text = pointer to an IntuiText structure.
PositiveText = pointer to an IntuiText structure.
NegativeText = pointer to an IntuiText structure.
PositiveFlags = flags for the IDCMP.
NegativeFlags = flags for the IDCMP.
Width, Height = the sizes required for the rendering of the requester.

RESULT

The return value is either TRUE or FALSE. See the text above for a complete description of the chain of events that might lead to either of these values being returned.

BUGS

None.

SEE ALSO

BuildSysRequestQ.

BeginRefresh

NAME

BeginRefresh — Sets up a window for optimized refreshing.

SYNOPSIS

BeginRefresh (Window) j
AO

FUNCTION

This routine sets up your window for optimized refreshing. It sets Intuition internal states and then sets up the layer underlying your window for a call to the layer library. There, the "clip rectangles" of the layer are reorganized in a fashion that causes any drawing performed in your window (until you call **EndRefresh())** to occur only in the regions that need to be refreshed. The term "clip rectangles" refers to the division of your window into visible and concealed rectangles. For more information about clipping rectangles and the layer library, **refer to the *Amiga ROM Kernel Manual.***

For instance, if you have a SIMPLE_REFRESH window that is partially concealed and the user brings it to the front, your program will receive a message asking it to refresh its display. If your program calls BeginRefresh() before doing any of the drawing, the layer that underlies your window will be arranged such that the only drawing that will actually take place will be that which goes to the newly revealed areas. This is very performance-efficient.

After your program has performed its refresh of the display, it should call **EndRefreshQ** to reset the state of the layer and the window. Then the program may proceed with drawing to the window as usual.

Your program learns that the window needs refreshing by receiving either a message of class REFRESHWINDOW through the IDCMP or an input event of class IECLASSJREFRESHWINDOW through the console device. Whenever the program is told that the window needs refreshing, it should call BeginRefresh() and EndRefreshQ to clear the refresh-needed state, even if no drawing will be done.

INPUTS

Window = pointer to the Window structure that needs refreshing.

RESULT

None.

BUGS

None.

SEE ALSO

EndRefreshQ.

NAME

BuildSysRequest — Builds and displays **a** system requester.

SYNOPSIS

```
BuildSysRequest(Window, BodyText, PositiveText, NegativeText,
                AO      Al        A2            A3
        IDCMPFlags, Width, Height);
        DO          Dl     D2
```

FUNCTION

This procedure builds **a** requester based on the supplied information. If all goes well and the requester is constructed, this procedure returns a pointer to the window in which the requester appears. That window will have the IDCMP **UserPort** and **WindowPort** initialized to reflect the flags found in the **IDCMPFlags** argument. The program may then Wait() on those ports to detect the user's response to your requester, which may include either selecting one of the gadgets or causing some other event to be noticed by Intuition (such as DISKINSERTED, for instance). After the requester is satisfied, your program should call the **FreeSysRequestQ** procedure to remove the requester and free up any allocated memory.

If it is not possible to construct the requester, this procedure will use the text arguments to construct a text string for a call to the DisplayAlert() procedure and then will return either TRUE or FALSE depending on whether **DisplayAlert()** returned FALSE or TRUE, respectively.

If the **Window** argument you supply is equal to NULL, a new window will be created for you in the Workbench screen. If you want the requester created by this routine to be bound to a particular window, you should not supply a **Window** argument of NULL.

The text arguments are used to construct the display. They are pointers to instances of the **IntuiText** structure.

The **BodyText** argument should be used to describe the nature of the requester. As usual with **IntuiText** data, you may link several lines of text together, and the text may be placed in various locations in the requester. This IntuiText pointer will be stored in the **ReqText** variable of the new requester.

The **PositiveText** argument describes the text that you want associated with the user choice of "Yes," "TRUE," "retry," or "good." If the requester is successfully opened, this text will be rendered in a gadget in the lower left of the requester; this gadget will have the **GadgetID** field set to TRUE. If the requester cannot be opened and the **Display**Alert() mechanism is used, this text will be rendered in the lower left corner of the alert display with additional text specifying that the left mouse button will select this choice. This pointer can be set to NULL, which specifies that there is no TRUE choice th&t can be made.

The **NegativeText** argument describes the text that you want associated with the user choice of "No," "FALSE," "cancel," or "bad." If the requester is successfully opened, this text will be rendered in a gadget in the lower right of the requester; this gadget will have the GadgetID field set to FALSE. If the requester cannot be opened and the DisplayAlertQ mechanism is used, this text will be rendered in the lower right corner of the alert display with additional text specifying that the right mouse button will select this choice. This pointer cannot be set to NULL. There must always be a way for the user to cancel this requester.

The positive and negative gadgets created by this routine have the following features:

o BOOLGADGET

o RELVERIFY

o REQGADGET

o TOGGLESELECT

When defining the text for your gadgets, you may find it convenient to use the special definitions used by Intuition for the construction of the gadgets. These definitions include AUTODRAWMODE, AUTOLEFTEDGE, AUTOTOPEDGE and AUTOFRONTPEN. You can find these in your local *intuition.h* (or ***intuition.i)*** **file.**

The **Width** and Height values describe the size of the requester. All of your **BodyText** must fit within the Width and Height of your requester. The gadgets will be created to conform to your sizes.

IMPORTANT NOTE: For the preliminary release of this procedure, a new window is opened in the same screen as the one containing your window. However, with a forthcoming update of Intuition this will change; the requester will be opened in the window supplied as an argument to this routine, if possible. The primary implication of this will be that the IDCMP flags and ports will be disturbed by a call to this routine. To assure upward compatibility, it is your responsibility to make sure that the ports and EDCMPFlags of the window passed to the routine are protected before the call to this routine.

INPUTS

Window = pointer to a Window structure.

BodyText = pointer to an **IntuiText** structure.
PositiveText = pointer to an **IntuiText** structure.
NegativeText = pointer to an **IntuiText** structure.
IDCMPFlags = the IDCMP flags you want used for the initialization of the IDCMP of the window containing this requester.
Width, Height = the size required to draw your requester.

RESULT

If the requester was successfully drawn in a window, the value returned by this procedure is **a** pointer to the window in which the requester was drawn. If, however, the requester cannot be drawn in the window, this routine will have called **Display Alert** () before returning and will pass back TRUE if the user pressed the left mouse button and FALSE if the user pressed the right mouse button.

BUGS

This procedure currently opens a window and then opens the requester within that window. Also, **if DisplayAlertQ** is called, the **PositiveText** and **NegativeText** are not rendered in the lower corners of the alert.

SEE ALSO

FreeSy sRequest ().
DisplayAlertQ.
ModifyIDCMP().
The Executive's **WaitQ** instruction.
AutoRequestQ.

NAME
ClearDMRequest - Clears the DMRequest of the window.

SYNOPSIS
ClearDMRequestfWindow);
AO

FUNCTION
Attempts to clear the DMRequester from the specified window. The DMRequester is the special requester that you attach to the double-click of the menu button; the user can then bring up that requester on demand. This routine will not clear the DMRequester if it is active (in use by the user). If you want to change the DMRequester after having called SetDMRequestQ, the correct way to start is by calling **ClearDMRequest()** until it returns **a** value of TRUE; then you can call **SetDMRequest()** with the new DMRequester.

INPUTS
Window = pointer to the structure of a window from which the DMRequest is to be cleared.

RESULT
If the DMRequest was not currently in use, this function zeroes out the DMRequest pointer in the window and returns TRUE.

If the DMRequest was currently in use, this function does not change the pointer and returns FALSE.

BUGS
None.

SEE ALSO
SetDMRequest().
Request ().

NAME

ClearMenuStrip — Clears the menu strip from the window.

SYNOPSIS

ClearMenuStrip (Window) j
AO

FUNCTION

Clears the menu strip from the window.

INPUTS

Window =s pointer to a **Window** structure.

RESULT

None.

BUGS

None.

SEE ALSO

SetMenuStripQ.

NAME

ClearPointer - Clears the pointer definition from *a window.

SYNOPSIS

ClearPointer(Window);
AO

FUNCTION

Clears the window of its own definition of the Intuition pointer. After **ClearPointerQ** is called, every time this window is active the default Intuition pointer will be the pointer displayed to the user. If your window is active when this routine is called, the change will take place immediately.

INPUTS

Window = pointer to the structure of the window to be cleared of its pointer definition.

RESULT

None.

BUGS

None.

SEE **ALSO**

SetPointerQ.

NAME

CloseScreen — Closes an Intuition screen.

SYNOPSIS

CloseScreen(Screen);
AO

FUNCTION

This function unlinks the screen, unlinks the VIewPort, and deallocates everything. It does not care whether or not there are still any windows attached to the screen and does not try to close any attached windows; in fact, it ignores them altogether. If this is the last screen, this function attempts to reopen Workbench.

INPUTS

Screen = pointer to the **Screen** structure to be cleared and deallocated.

RESULT

None.

BUGS

None.

SEE **ALSO**

OpenScreenQ.

NAME

CloseWindow — Closes an Intuition window. «»

SYNOPSIS

```
CloseWindow(Window);
            AO
```

FUNCTION

This function closes an Intuition window. It unlinks it from the system, unallocates its memory, and, if its screen is a system one that would be empty without the window, closes the system screen, too.

A grim, foreboding note: if you are ever rude enough to **CloseWindowQ** on a window that has an IDCMP without first having Reply()'d to all of the messages to the IDCMP port, Intuition in turn will be so rude as to reclaim and deallocate its messages without waiting for your permission.

Another grim note: if you have added a menu strip to this window (via a call to SetMenuStripO) you must be sure to remove that menu strip (via a call to ClearMenuStrip()) before closing your window. **CloseWindow**() does not check whether the menus of your window are currently being used when the window is closed. If this happens to be the case, as soon as the user releases the menu button the system will crash with pyrotechnics that are usually quite lovely.

INPUTS

Window = a pointer to **a Window** structure.

RESULT

None.

BUGS

None.

SEE ALSO

OpenWindow().
ClaseScreenQ.

NAME

CloseWorkBench — Closes the Workbench screen.

SYNOPSIS

BOOL **CloseWorkBenchQ;**

FUNCTION

This routine attempts to close the Workbench. If the Workbench is open, it tests whether or not any applications have opened windows on the Workbench and returns FALSE if so. Otherwise, it cleans up all special buffers, closes the Workbench screen, makes the Workbench program mostly inactive (it will still monitor disk activity), and returns TRUE.

If the Workbench screen isn't open when this routine is called, TRUE is returned immediately.

INPUTS

None.

RESULT

TRUE if the Workbench screen is closed.
FALSE if anything went wrong and the Workbench screen is still out there.

BUGS

None.

SEE ALSO

None.

NAME
CurrentTime — Gets the current time values.

SYNOPSIS
ULONG Seconds, Micros;
CurrentTime(&Seconds, & Micros);
DO Dl

FUNCTION
This function puts copies of the current time into the supplied argument pointers. This time value is not extremely accurate, nor is it of a very fine resolution. The time will be updated no more than sixty times a second and will typically be updated far fewer times a second.

INPUTS
Seconds = pointer to a ULONG variable to receive the current seconds value.
Micros = pointer to a ULONG variable for the current microseconds value.

RESULT
Puts the time values into the memory locations specified by the arguments.

BUGS
None.

SEE ALSO
None.

NAME

DisplayAlert — Creates a display of an alert message.

SYNOPSIS

```
DisplayAlert(AlertNumber, String, Height);
             DO           AO      D1
```

FUNCTION

Creates an alert display with the specified message.

If the system can recover from this alert, it is a RECOVERY_ALERT. The routine waits until the user presses one of the mouse buttons, after which the display is restored to its original state and a BOOL value is returned by this routine to specify whether or not the user pressed the left mouse button.

If the system cannot recover from this alert, it is a DEADEND_ALERT, and this routine returns immediately upon creating the alert display. The return value is FALSE.

The **AlertNumber** is a LONG value, related to the value sent to the AlertQ routine. The only bits that are pertinent to this routine, however, are the ALERTJTYPE bits. These bits must be set to RECOVERY_ALERT for alerts from which the system may safely recover or DEADEND_ALERT for fatal alerts. These states are described in the paragraph above. A third type of alert, the DAISY_ALERT, is used only by the Executive.

The **String** argument points to an AlertMessage string. The AlertMessage string is composed of one or more substrings, each of which contains the following components:

o First, a 16-bit x coordinate and an 8-bit y coordinate, describing where on the alert display you want this string to appear. The y coordinate describes the offset to the baseline of the text.

o Then, the bytes of the string itself, which must be null-terminated (end with a byte of zero).

o Lastly, the continuation byte, which specifies whether or not another substring follows this one. If the continuation byte is non-zero, there is another substring to be processed in this AlertMessage. If the continuation byte is zero, this is the last substring in the message.

The last argument, **Height,** describes how many video lines tall you want the alert display to be.

INPUTS

AlertNumber = the number of this AlertMessage. <The only pertinent bits of this number are the ALERTJTYPE bits. The rest of the number is ignored by this routine.

String = pointer to the alert message string, as described above.

Height = minimum display lines required for your message.

RESULT

A BOOL value of TRUE or FALSE. If this is a DEADEND_ALERT, FALSE is always the return value. If this is a RECOVERY_ALERT, the return value will be TRUE if the user presses the left mouse button in response to your message and FALSE if the user presses the right button.

BUGS

If the system is in more trouble than you think, the level of your alert may become DEADEND_ALERT without you ever knowing about it.

SEE ALSO

None.

NAMET

DisplayBeep — "Beeps" the video display.

SYNOPSIS

DisplayBeep(Screen);
A0

FUNCTION

"Beeps" the video display by flashing the background color of the specified screen. If the **Screen** argument is NULL, every screen in the display will be beeped. Flashing all screens is not **a** polite thing to do, so this should be reserved for dire circumstances.

Such a routine is supported because the Amiga has no internal bell or speaker. When the user needs to know of an event that is not serious enough to require the use of a requester, the DisplayBeep() function should be called.

INPUTS

Screen = pointer to a **Screen** structure. If NULL, *every* Intuition screen will be flashed.

RESULT

None.

BUGS

None.

SEE **ALSO**

None.

NAME

DoubleClick — Tests two time values for double-click timing.

SYNOPSIS

```
DoubleClick(StartSeconds, StartMicros, CurrentSeconds,
            DO            Dl           D2
        CurrentMicros)j
            D3
```

FUNCTION

Compares the difference in the time values with the double-click timeout range that the user (using the Preferences tool or some other source) has configured into the system. If the difference between the specified time values is within the current double-click time range, this function returns TRUE; otherwise, it returns FALSE.

These time values can be found in **InputEvents** and IDCMP messages. The time values are not perfect; however, they are precise enough for nearly all applications.

INPUTS

StartSeconds, StartMicros = the timestamp value describing the start of the double-click time period you are considering.

CurrentSeconds, CurrentMicros = the timestamp value describing the end of the double-click time period you are considering.

RESULT

If the difference between the supplied timestamp values is within the double-click time range in the current set of Preferences, this function returns TRUE; otherwise, it returns FALSE.

BUGS

None.

SEE ALSO

CurrentTimeQ.

NAME

DrawBorder — Draws the specified border into the RastPort.

SYNOPSIS

```
DrawBorder(RastPort, Border, LeftOffset, TopOffset);
           AO        Al      DO          Dl
```

FUNCTION

First, this function sets up the drawing mode and pens in the **RastPort** according to the arguments of the **Border** structure. Then, it draws the vectors of the **Border** argument into the **RastPort,** offset by the LeftOffset and TopOffset. This routine does Intuition window clipping as appropriate—if you draw a line outside of your window, your imagery will be clipped at the window's edge.

If the **NextBorder** field of the **Border** argument is non-zero, the next **Border** is rendered as well (return to the top of this **FUNCTION** section for details).

INPUTS

RastPort = pointer to the **RastPort** to receive the border crossing.
Border = pointer to a **Border** structure.
LeftOffset = the offset that will be added to each vector's x coordinate.
TopOffset = the offset that will be added to each vector's y coordinate.

RESULT

None.

BUGS

None.

SEE ALSO

None.

NAME

Drawlmage -- Draws the specified Image into the RastPort.

SYNOPSIS

```
DrawImage(RastPort, Image, LeftOffset, TopOffset);
          AO        Al     DO          Dl
```

FUNCTION

First, this function sets up the drawing mode and pens in the **RastPort** according to the arguments of the **Image** structure. Then, it moves the image data of the **Image** argument into the **RastPort,** offset by the **LeftOffset** and **TopOffset.** This routine does Intuition window clipping as appropriate—if you draw an image outside of your window, your imagery will be clipped at the window's edge.

-If the **Nextlmage** field of the **Image** argument is non-zero, the next **Image** is rendered as well (return to the top of this section for details).

INPUTS

RastPort = pointer to the **RastPort** to receive the border crossing.
Image = pointer to an **Image** structure.
LeftOffset = the offset that will be added to the Image's x coordinate.
TopOffset = the offset that will be added to the Image's y coordinate.

RESULT

None.

BUGS

None.

SEE ALSO

None.

NAME

EndRefresh — Ends the optimized refresh state of the window.

SYNOPSIS

```
EndRefreshfWindow, Complete);
                   AO       DO
```

FUNCTION

This function gets you out of the special refresh state of your window. It is called following a call to BeginRefresh(), which begins the special refresh state. While your window is in the refresh state, the only drawing that will be wrought in your window will be to those areas that were recently revealed and that need to be refreshed.

After your program has done all the needed refreshing for this window, this routine is called to restore the window to its non-refreshing state. Then all rendering will go to the entire window as usual.

The **Complete** argument is a Boolean TRUE or FALSE value used to describe whether or not the refreshing that has been done is all that needs to be done at this time. Most often, this argument will be TRUE. However, if, for instance, you have multiple tasks or multiple procedure calls that must run to completely refresh the window, each can call its own Begin/EndRefreshQ pair with a **Complete** argument of FALSE, and only the last calls with a Complete argument of TRUE.

INPUTS

Window = pointer to the **Window** currently in optimized-refresh mode.
Complete = Boolean TRUE or FALSE describing whether or not this window is completely refreshed.

RESULT

None.

BUGS

None.

SEE ALSO

BeginRefreshQ.

NAME

EndRequest — Ends the request and resets the window.

SYNOPSIS

```
EndRequest(Requester, Window)
           A0         A1
```

FUNCTION

This function ends the request by erasing the requester and resetting the window. Note that this does not necessarily clear all requesters from the window, only the specified one. If the window labors under other requesters, they will remain in the window.

INPUTS

Requester = pointer to the structure of the requester to be removed.
Window = pointer to the **Window** structure with which this requester is associated.

RESULT

None.

BUGS

None.

SEE ALSO

None.

NAME

FreeRemember -- Frees the memory allocated by calls to AllocRememberQ.

SYNOPSIS

```
FreeRemember(RememberKey, Really Forget);
                          AO          DO
```

FUNCTION

This function frees up memory allocated by the **AllocRemember()** function. It will free up just the **Remember** structures, which supply the link nodes that tie your allocations together, or it will deallocate both the link nodes and your memory buffers.

If you want to deallocate just the **Remember** structure link nodes, you should set the **ReallyForget** argument to FALSE. However, if you want **FreeRememberQ** to really forget about all the memory, including both the **Remember** structure link nodes and the buffers you requested via earlier calls to **AllocRemember(),** you should set the ReallyForget argument to TRUE. If you're not sure whether or not you want to **ReallyForget,** refer to figure **11-1.**

INPUTS

RememberKey = the address of a pointer to a **Remember** structure. This pointer should either be NULL or be set to some value (possibly NULL) by **a** call to **AllocRemember().** For example:

```
struct Remember *RememberKey;
RememberKey = NULL;
AllocRememberf&RememberKey, BUFSIZE, MEMF_CHIP);
FreeRemember(&RememberKey, TRUE);
```

ReallyForget = a BOOL FALSE or TRUE describing, respectively, whether you want to free up only the **Remember** nodes or whether you want this procedure to really forget about **all** of the memory, including both the nodes and the memory buffers pointed to by the nodes.

RESULT

None.

BUGS

None.

SEE ALSO

AllocRememberQ.

NAME

FreeSysRequest - Frees up memory used by a call to BuildSysRequestQ.

SYNOPSIS

FreeSysRequest(Window);
A0

FUNCTION

This routine frees up all memory allocated by a successful call to the **BuildSysRequestQ** procedure. If **Build**SysRequest() returned a pointer to a **Window** structure, then your program can Wait() for the message port of that window to detect an event that satisfies the requester. When you want to remove the requester, you call this procedure. It ends the requester and deallocates any memory used in the creation of the requester.

NOTE: If Build SysRequestQ did not return a pointer to a window, you should not call **FreeSysRequestQ.**

INPUTS

Window = a copy of the window pointer returned by a successful call to the **BuildSysRequestQ** procedure.

RESULT

None.

BUGS

None.

SEE ALSO

BuildSysRequestQ.
The Executive's **WaitQ** instruction.
AutoRequest Q.

NAME

GetDefPrefs — Gets a copy of the Intuition default Preferences.

SYNOPSIS

GetDefPrefs(PrefBuffer, Size);
AO DO

FUNCTION

This function gets a copy of the Intuition default Preferences data. It writes the data into the buffer you specify. The number of bytes you want copied is specified by the Size argument.

The default Preferences are those that Intuition uses when it is first opened. If no Preferences file is found, these are the preferences that are used. These would also be the start-up Preferences in an environment that does not use AmigaDOS.

It is legal to take a partial copy of the Preferences structure. The more pertinent Preferences variables have been grouped near the top of the structure to facilitate the memory conservation that can be had by taking a copy of only some of the Preferences structure.

INPUTS!

PrefBuffer = pointer to the memory buffer to receive your copy of the Intuition Preferences.

Size = the number of bytes in your **PrefBuffer**—the number of bytes you want copied from the system's internal Preference settings.

RESULT

Returns your Preferences pointer.

BUGS

No^e.

SEE ALSO

GetPrefsQ.

NAME

GetPrefs — Gets the current setting of the Intuition Preferences.

SYNOPSIS I

```
GetPrefs(PrefBuffer, Size);
        |      AO      DO
```

FUNCTION

This function gets a copy of the current Intuition Preferences data and writes the data into the buffer you specify. The number of bytes you want copied is specified by the Size argument.

It is legal to take a partial copy of the Preferences structure. The more pertinent Preferences variables have been grouped near the top of the structure to facilitate the memory conservation that can be had by taking a copy of only some of the Preferences structure.

INPUTS

PrefBuffer = pointer to the memory buffer to receive your copy of the Intuition Preferences.

Size = the number of bytes in your **PrefBuffer**—the number of bytes you want copied from the system's internal Preference settings.

RESULT

Returns a copy of your Preferences pointer.

BUGS

None.

SEE ALSO

GetDefPrefsQ.

NAME

InitRequester — Initializes a Requester structure.

SYNOPSIS

InitRequester(Requester);
AO

FUNCTION

The original text for this function was:

> This function initializes a requester for general use. After calling **InitRequester()**, you need fill in only those requester values that fit your needs. The other values are set to states that Intuition regards as NULL.

All this routine actually does is fill the specified **Requester** structure with zeros. There is no requirement to call this routine before using a **Requester** structure. For the sake of backward compatibility, this function call remains, but its sole effect is, and is guaranteed to always be, a zero, a mystery, an enigma.

INPUTS

Requester = a pointer to a **Requester** structure.

RESULT

None.

BUGS

None.

SEE ALSO

None.

NAME

IntuiTextLength - Returns the length (pixel width) of an IntuiText.

SYNOPSIS

IntuiTtextLength(IText);
AO

FUNCTION

This rputine accepts a pointer to an instance **of** an **IntuiText** structure and return^ the length (the pixel width) of the string that is represented by that instance of the structure.

All of ^he usual **IntuiText** rules apply. Most notably, if the **Font** pointer of the structure is set to NULL, you will get the pixel width of your text in terms of the current default font.

INPUTS

IText = pointer to an instance of an **IntuiText** structure.

RESULT

Returns the pixel width of the text specified **by** the **IntuiText** data.

BUGS

None.

SEE ALSO

None.

NAME

ItemAddress — Returns the address of the specified Menultem.

SYNOPSES

```
ItemAddress(MenuStrip, MenuNumber);
                AO          DO
```

FUNCTION

Th^s routine feels through the specified **MenuStrip** and returns the address of the item specified by the **MenuNumber.** Typically, you will use this routine to get the address of a **Menultem** from a **MenuNumber** sent to you by Intuition after the user has played with your menus.

This routine requires that the arguments be well defined. **MenuNumber** may be equal to MENUNULL, in which case this routine returns NULL. If **MenuNumber** does not equal MENUNULL, it is presumed to be a valid item number selector for your **MenuStrip,** which includes a valid menu number and a valid item number. If the item specified by the above two components has a subitem, the **MenuNumber** may have a subitem component too.

Note that there must be both a menu number and an item number. Because a subitem specifier is optional, the address returned by this routine may point to either an item or a subitem.

INPUTS

MenuStrip = a pointer to the first menu in your menu strip.
MenuNumber = the value that contains the packed data that selects the menu and item (and subitem).

RESULT

If **MenuNumber** == MENUNULL, this routine returns NULL. Otherwise, this routine returns the address of the **Menultem** specified by **MenuNumber.**

BUGS

None.

SEE ALSO

The "Menus" chapter in this book (chapter 6) for more information about **Menu Numbers.**

NAME

MakeScreen — Does an Intuition-integrated **MakeVPort()** of a custom screen.

SYNOPSIS

```
MakeScreen(Screen);
           AO
```

FUNCTION

This procedure allows you to do a **MakeVPort()** for the **ViewPort** of your custom screen in an Intuition-integrated way. This allows you to do your own screen njianipulations without worrying about interference with Intuition's usage of the same **ViewPort.**

After calling this routine, you can call **RethinkDisplay()** to incorporate the new **ViewPort** of your custom screen into the Intuition display.

INPUTS

Screen = address of the **Screen structure.**

RESULT

None.

BUGS

None.

SEE ALSO

RethinkDisplayQ.
RemakeDisplay().
The graphics library's **MakeVPortQ.**

NAME

MddifylDCMP - Modifies the state of the window's IDCMP.

SYNOPSIS

MJ>difyEDCMP(Window, IDCMPFlags);
I **AO** **DO**

FUNCTION

TMs routine modifies the state of your window's IDCMP (Intuition Direct Com-minication Message Port). The state is modified to reflect your desires as described by the flag bits in the value IDCMPFlags. If the IDCMPFlags argument equals NULL, you are asking for the ports to be closed; if they are op^n, they will be closed. If you set any of the **IDCMPFlags,** this means that **yoii** want the message ports to be open; if not currently open, the ports will be **opened.**

The four actions that might be taken are described below:

- If there is currently no IDCMP in the given window and IDCMPFlags is NULL, nothing happens.
- If there is currently no IDCMP in the given window and any of the **IDCMPFlags** are selected (set), the IDCMP of the window is created, including allocating and initializing the message ports and allocating a signal Ibit for your port. See "Input and Output Methods" (chapter 8) for full details.
- If the IDCMP for the given window is opened and the IDCMPFlags argument is NULL, Intuition will close the ports, free the buffers, and free your signal bit. The current task must be the same one that was active when this signal bit was allocated.
- If the IDCMP for the given window is opened and the IDCMPFlags argu-jment is not NULL, this means that you want to change which events will be broadcast to your program through the IDCMP.

NQTE: You can set up the Window->UserPort to any port of your own before you call **ModifyIDCMP().** If **IDCMPFlags** is non-null but your **Us^rPort** is already initialized, Intuition will assume that it is a valid port with tas^c and signal data preset and will not disturb your set-up; Intuition will just allocate the Intuition message port for your window. The converse is true as well; if UserPort is NULL when you call here with IDCMPFlags == NULL, only the Intuition port will be deallocated. This allows you to use a port that yoU already have allocated:

- **Open{Window()** with **IDCMPFlags** equal to NULL (open no ports).
- Set tlje **UserPort** variable of your window to any valid port of your own choosing.
- Call **fyIodifyIDCMP()** with **IDCMPFlags** set to what you want.
- Then,; to clean up later, set **UserPort** equal to NULL before calling **CloseWindow()** (leave **IDCMPFlags** alone).

A grim, foreboding note: If you are ever rude enough to close an IDCMP without first having **ReplyQ'd** to all of the messages sent to the IDCMP port, Intuition!will in turn be so rude as to reclaim and deallocate its messages without waiting for your permission.

INPUTS

Window = pointer to the **Window** structure containing the IDCMP ports.
IDCMPFlags = the flag bits describing the new desired state of the **IDCMP.**

RESULT

None.

BUGS

None.

SEE ALSO

OpenW|ndow().

NAME

ModifyProp — Modifies the current parameters of a proportional gadget.

SYNOPSIS

```
ModifyProp(PropGadget, Pointer, Requester,
                 AO           Al         A2
    Flags, HorizPot, VertPot, HorizBody, VertBody);
     DO      Dl        D2        D3          D4
```

FUNCTION

This routine modifies the parameters of the specified proportional gadget. The gadget's internal state is then recalculated and the imagery is redisplayed.

The **Pointer** argument can point to either a **Window** or a Screen structure. Which one it actually points to is decided by examining the SCRGADGET flag of the gadget. If the flag is set, **Pointer** points to a **Screen** structure; otherwise, it points to a **Window** structure.

The **Requester** variable can point to a Requester structure. If the gadget has the REQGADGET flag set, the gadget is in a requester and the **Pointer** must necessarily point to a window. If this is not the gadget of a requester, the **Requester** argument may be NULL.

INPUTS

PropGadget = pointer to the structure of **a** proportional gadget.
Pointer = pointer to the structure of the "owning" display element of the gadget, which is **a** window or **a** screen.
Requester = pointer to a **Requester** structure (this may be NULL if this is not **a** requester gadget).
Flags = value to be stored in the **Flags** variable of the **PropInfo.**
HorizPot = value to be stored **in** the **HorizPot** variable **of** the **PropInfo.**
VertPot = value to be stored in the **VertPot** variable of the **PropInfo,**
HorizBody = value to be stored in the **HorizBody** variable of the **PropInfo.**
VertBody = value to be stored in the **VertBody** variable of the **PropInfo.**

RESULT

None.

BUGS

None.

SEE ALSO

None.

NAME

MovoScreen — Attempts to move the screen by the delta amounts.

SYNOPSIS

MoveScreen(Screen, DeltaX, DeltaY);
AO DO Dl

FUNCTION

Attempts to move the specified screen. This movement must follow one constraint (only for the current release of the software): horizontal movements are ignored.

If th£ **DeltaX** and **DeltaY** variables you specify would move the screen in a way that violates the above restriction, the screen will be moved as far as possible.

INPUTS

Screten = pointer to a **Screen** structure.
DeltaX = amount to move the screen on the x axis.
DeltaY = amount to move the screen on the y axis.

RESULT I

None|.

BUGS

None.

SEE ALS(

None.

NAMEJ

tyfoveWindow — Ask Intuition to move a window.

SYNOPSIS

```
fy[oveWindow(Window, DeltaX, DeltaY);
             AO      DO      Dl
```

FUNCTION

This routine sends a request to Intuition asking to move the window the specified dijstance. The delta arguments describe how far to move the window along the respective axes. Note that the window will not be moved immediately; it will be mpved the next time Intuition receives an input event, which happens currently at **a** minimum rate of ten times per second and **a** maximum of sixty times **a** second.

Tljis routine does no error-checking. If your delta values specify some far corner of jthe universe, Intuition will attempt to move your window to the far corners of the universe. Because of the distortions in the space-time continuum that can result from this, as predicted by special relativity, the result is generally not a pretty sight.

INPUTS[1]

Window = pointer to the structure of the window to be moved.
DeltaX = signed value describing how far to move the window on the x axis.
DeltaY = signed value describing how far to move the window on the y axis.

RESULT|

None.

BUGS

Nonje.

SEE ALSb

SizefWindowQ.
WiiJdowToFront().
WiddowToBackQ-

NAME
    OffGadget — Disables the specified gadget.

SYNOPSIS
```
    OffGadget(Gadget, Pointer, Requester);
              AO      Al       A2
```

FUNCTION
    This Command disables the specified gadget. When a gadget is disabled, these things happen:

- o Its imagery is displayed ghosted.
- o The GADGDISABLED flag is set.
- o The gadget cannot be selected by the user.

The Pointer argument must point to a Window structure. The Requester variable can point to a Requester structure. If the gadget has the REQGADGET flag set, the gadget is in a requester and Pointer must necessarily point to the window containing that requester. If this is not the gadget of a requester, the Requester argument may be NULL.

NOTE: It is never safe to tinker with the gadget list yourself. Do not supply some gadget that Intuition has not already processed in the usual way.

NOTE: If you have specified that this is a gadget of a requester, that requester must be currently displayed.

INPUTS
    Gadget = pointer to the structure of the gadget that you want disabled.
    Pointer = pointer to a Window structure.
    Requester = pointer to a Requester structure (may be NULL if this is not a
        requester gadget list).

RESULT
    None.

BUGS
    None.

SEE ALSO
    OnGadget().

NAME

Offl^lenu — Disables the given menu or menu item.

SYNOPSIS

```
Ofl]Menu(W'mdow, MenuNumber);
              AO          DO
```

FUNCTION

This command disables a subitem, an item, or a whole menu. If the base of the menu number matches the menu currently revealed, the menu strip is **redisplayed.**

INPUTS h

Window = pointer to the **Window** structure.
MenuNumber = the menu piece to be enabled.

RESULT

Nonfc.

BUGS

Nonp.

SEE ALSb

On^IenuQ,

NAME

OnGadget — Enables the specified gadget.

SYNOPSIS

```
OnGadget(Gadget, Pointer, Requester);
         AO      A1       A2
```

FUNCTION

This command enables the specified gadget. When a gadget is enabled, these things happen:

- o Its imagery is displayed normally (not ghosted).
- o The GADGDISABLED flag is cleared.
- o The gadget can thereafter be selected by the user.

The Pointer argument must point to a Window structure. The Requester variable can point to a Requester structure. If the gadget has the REQGADGET flag set, the gadget is in a requester and Pointer must point to the Window containing the requester. If this is not the gadget of a requester, the requester argument may be NULL.

NOTE: It is never safe to tinker with the gadget list yourself. Do not supply some gadget that Intuition has not already processed in the usual way.

NOTE:: If you have specified that this is a gadget of a requester, that requester must be currently displayed.

INPUTS

Gadget = pointer to the structure of the gadget that you want enabled.
Pointer = pointer to a Window structure.
Requester = pointer to a Requester structure (may be NULL if this is not a requester gadget list).

RESULT

None.

BUGS

None.

SEE ALSO

OffG^dgetQ.

NAME

OnMenu -- Enables the given menu or menu item.

SYNOPSIS

```
OpMenu(Window, MenuNumber);
  !          AO        DO
```

FUNCTION

This command enables a subitem, an item, or a whole menu. If the base of the menu number matches the menu currently revealed, the menu strip is redisplayed.

INPUTS!

Window = pointer to the window.
MenuNumber = the menu piece to be enabled.

RESULT

Nope.

BUGS

None.

SEE ALSO

OffMenuQ.

NAME

OpenScreen — Opens an Intuition screen.

SYNOPSIS

OpenScreen(NewScreen)
A0

where the **NewScreen** structure is initialized with:

Left, Top, Width, Height, Depth, DetailPen, BlockPen, ViewModes, Type, Font, DefaultTitle, Gadgets

FUNCTION

This command opens an Intuition screen according to the specified parameters. It does all the allocations, sets up the screen structure and all substructures completely, and links this screen's **ViewPort** into Intuition's **View** of the world.

Before you call **OpenScreen(),** you must initialize an instance of a **NewScreen** structure. **NewScreen** is **a** structure that contains all of the arguments needed to open a screen. The **NewScreen** structure may be discarded immediately after it is used to open the screen.

The **TextAttr** pointer that you supply as an argument will be used as the default font for all Intuition-managed text that appears in the screen and its windows. This includes, but is not limited to, the text on the title bars of both the screen and windows.

The SHOWTITLE flag is set to TRUE by default when a screen is opened. This causes the screen's title bar to be displayed when the screen first opens. To hide the title bar, you must call the routine **ShowTitle().**

INPUTS

NewScreen = pointer to an instance of a **NewScreen** structure, which is initialized with the following information:

LeftEdge = initial x position of your screen (should be zero for now).
TopEdge = initial y position of the opening screen.
Width = the width for this screen's **RastPort.**
Height = the height for this screen's **RastPort.**
Depth = number of bit-planes.
DetailPen == pen number for details (such as gadgets or text in the title bar).
BlockPen = pen number for block fills (such as the title bar).
Type = screen type (for any screen not created by Intuition, this should be equal to CUSTOMSCREEN). Types currently supported include only CUSTOMSCREEN, which is your own screen.

You may also set the **Type** flag CUSTOMBITMAP and then supply your own BitMap for Intuition to use, rather than having Intuition allocate the display memory for you.

ViewModes = the appropriate flags for the data type **ViewPort.Modes.** These might include:

HIRES for this screen to be HIRES width.
INTERLACE for the display to switch to interlaced mode.
SPRITES for this screen to use sprites.
DUALPF for dual-playfield mode.

Font = pointer to the default **TextAttr** structure for this screen and all windows that open in this screen.

DefaultTitle = pointer to a line of text that will be displayed along the screen's title bar. The text will be null-terminated. If this argument is set to NULL, no text will be produced.

Gadgets = this should be set to NULL.

GustomBitMap = If you're not supplying **a** custom **BitMap,** this value is ignored. However, if you have your own display memory that you want used for this screen, the **CustomBitMap** argument should point to the **BitMap** that describes your display memory. See the "Screens" chapter and the *Amiga ROM Kernel Manual* for more information about **BitMaps.**

RESULT

If all is well, the routine returns the pointer to your new screen.
If anything goes wrong, the routine returns NULL.

BUGS

None.

SEE ALSO

OpenWindowQ.
ShowTitleQ.

NAME

OpenWindow — Opens an Intuition window.

SYNOPSIS

OpenWindow(NewWindow);
A0

where the **NewWindow** structure is initialized with:

Left, Top, Width, Height, DetailPen, BlockPen, Flags, IDCMPFlags, Gadgets, CheckMark, Text, Type, Screen, BitMap, MinWidth, MinHeight, Max Width, MaxHeight

FUNCTION

This command opens an Intuition window of the given height, width, and depth, including the specified system gadgets as well as any of your own. It allocates everything you need to get going.

Before you call **OpenWindowQj** you must initialize an instance of a **NewWindow** structure, which contains all of the arguments needed to open a window. The **NewWindow** structure may be discarded immediately after it is used to open the window.

If **Type** == CUSTOMSCREEN, you must have opened your own screen already via a call to **OpenScreen().** Then Intuition uses your **Screen** argument for the pertinent information needed to get your window going. On the other hand, if **Type** == one of Intuition's standard screens, your **Screen** argument is ignored. Instead, Intuition will check to see whether or not that screen already exists; if it does not, it will be opened first before Intuition opens your window in the standard screen. If the flag SUPER_BITMAP is set, the **BitMap** variable must point to your own **BitMap.** The **DetailPen** and the **BlockPen** are used for system drawing; for instance, the title bar is first filled using the **BlockPen,** and then the gadgets and text are drawn using **DetailPen.** You can supply special pens for your window, or you can use the screen's pens instead (by setting either of these arguments to -1).

INPUTS

NewWindow = pointer to an instance of a **NewWindow** structure, which is initialized with the following data:

LeftEdge = the initial x position for your window.
TopEdge = the initial **y** position for your window.
Width = the initial width of this window.

Height = the initial height of this window.
DetailPen = pen number (or -1) for the drawing of window details (such as gadgets or text in the title bar).
BlockPen = pen number (or -1) for window block fills (such as the title bar)
Flags == specifiers for your requirements of this window, as follows.

- **o** System gadgets you want attached to your window:
  - **o** WINDOWDRAG allows this window to be dragged.
  - **o** WINDOWDEPTH lets the user depth-arrange this window.
  - **o** WINDOWCLOSE attaches the standard close gadget.
  - **o** WINDOWSIZING allows this window to be sized. If you ask for the WINDOWSIZING gadget, you must specify one or both of the flags SIZEBRIGHT and SIZEBBOTTOM below; if you do not, the default is SIZEBRIGHT. See the following SIZEBRIGHT and SIZEBBOTTOM items for extra information.
  - **o** SIZEBRIGHT is **a** special system gadget flag that you set to specify whether or not you want the right border adjusted to account for the physical size of the sizing gadget. The sizing gadget must, after all, take up room in either the right or the bottom border (or both, if you like) of the window. Setting either this or the SIZEBBOTTOM flag selects which edge will take up the slack. This will be particularly useful to applications that want to use the extra space for other gadgets (such as a proportional gadget and two Booleans done up to look like scroll bars) or, for instance, applications that want every possible horizontal bit and are willing to lose lines vertically.

    NOTE: If you select WINDOWSIZING, you must select either SIZEBRIGHT or SIZEBBOTTOM or both. If you select neither, the default is SIZEBRIGHT.
  - **o** SIZEBBOTTOM is **a** special system gadget flag that you set to specify whether or not you want the bottom border adjusted to account for the physical size of the sizing gadget. For details, refer to SIZEBRIGHT above. NOTE: If you select WINDOWSIZING, you must select either SIZEBRIGHT or SIZEBBOTTOM or both. If you select neither, the default is SIZEBRIGHT.

- o GIMMEZEROZERO produces easy but expensive output.
- **o** Type of window raster you want:
  - **o** SIMPLEJREFRESH
  - **o** SMARTJREFRESH
  - **o** SUPER_BITMAP

- BACKDROP specifies whether or not you want this window to be one of Intuition's special backdrop windows. See BORDERLESS as well.
- REPORTMOUSE specifies whether or not you want the program to "listen" to mouse movement events whenever its window is active. If you want to change whether or not your window is listening to the mouse after you have opened your window, you can call ReportMouse(). Whether or not your window is listening to the mouse is also affected by gadgets, because they can cause the program to get mouse movement reports. The reports (either InputEvents or messages on the IDCMP) that you get will have the x,y coordinates of the current mouse position, relative to the upper left corner of your window (GIMMEZEROZERO notwithstanding). This flag can work in conjunction with the IDCMP flag called MOUSEMOVE, which allows your program to listen via the IDCMP.
- BORDERLESS should be set if you want a window with no default border padding. Your window may have border padding anyway, depending on the gadgetry you have requested for the window, but you will not get the standard border lines and spacing that come with typical windows. This is a good way to take over the entire screen, since you can have a window cover the entire width of the screen using this flag. This will work particularly well in conjunction with the BACKDROP flag (see above), because it allows you to open a window that fills the entire screen.

  NOTE: This is not a flag that you want to set casually, since it may cause visual confusion on the screen. The window borders are the only dependable visual division between various windows and the background screen. Taking away the border takes away that visual cue, so make sure that your design does not need it before you proceed.
- ACTIVATE is the flag you set if you want this window to automatically become the active window. The active window is the one that receives input from the keyboard and mouse. It is usually a good idea to have the window you open when your application first starts up be an ACTIVATED one, but all others opened later should not be ACTIVATED. (If the user is off doing something with another screen, for instance, your new^ window will change where the input is going, which would have the effect of yanking the input rug from under the user.) Please use this flag thoughtfully and carefully.
- RMBTRAP, when set, causes the right mouse button events to be trapped and broadcast as events. Your program can receive these events through either the IDCMP or the console.

IDCMPFlags = IDCMP is the acronym for Intuition Direct Communications Message Port. It is Intuition's sole acronym, given in honor of all hackheads who love to mangle our brains with maniacal names; fashioned especially cryptic and unpronounceable to make them squirm with sardonic delight. Here's to you, my chums. Meanwhile, I still opt (and argue) for simplicity and elegance.

If any of the IDCMP flags is selected, Intuition will create a pair of message ports and use them for direct communications with the task that is opening this window (as compared with broadcasting information via the console device). See the "Input and Output Methods" chapter of this book (chapter 8) for complete details.

You request an IDCMP by setting any of these flags. Except for the special "verify" flags, every other flag you set tells Intuition that if a given event occurs that your program wants to know about, Intuition should broadcast the details of that event through the IDCMP rather than via the console device. This allows a program to interface with Intuition directly, rather than going through the console device.

Remember, if you are going to open both an IDCMP and a console, it will be far better to get most of the event messages via the console. Reserve your usage of the IDCMP for special performance cases; that is, when you are not going to open a console for your window and yet you do want to learn about a certain set of events (for instance, CLOSEWINDOW); another example is SIZEVERIFY, which is **a** function that you get only through the use of the IDCMP (because the console does not give you any way to talk to Intuition directly).

On the other hand, if the **IDCMPFlags** argument is equal to zero, no IDCMP is created and the only way you can learn about any window event for this window is via a console opened for this window. For instance, you have no way to SIZEVERIFY.

If you want to change the state of the IDCMP after you have opened the window (including opening or closing the IDCMP), you call the routine **ModifyIDCMP().**

The flags you can set are explained below:

- REQVERIFY is **a** flag that, like SIZEVERIFY and MENUVERIFY (see below), specifies that you want to make sure that your graphical state is quiescent before something extraordinary happens, such as the drawing of a rectangle of graphical data in your window. If you are drawing in that window, you probably will wish to make sure that you have ceased drawing before the user is allowed to bring up the DMRequest you have set up. The same goes for when the system has a requester for the user. Set this flag to ask for that verification step.
- REQCLEAR is the flag you sett to get notification when the last

requester is cleared from your window and it is safe for you to start output again (presuming that you are using REQVERIFY).

- REQSET is a flag that you set to receive a broadcast when the first requester is opened in your window. Compare this with REQCLEAR above. This function is distinct from REQVERIFY. REQSET merely tells your program that a requester has opened, whereas REQVERIFY requires the program to respond before the requester is opened.
- MENUVERIFY is the flag you set to have Intuition stop and wait for your program to finish all graphical output to the window before drawing the menus. Menus are currently drawn in the most memory* efficient way, which involves interrupting output to all windows in the screen before the menus are drawn. If you need to finish your graphical output before this happens, you can set this flag to make sure that you do.
- SIZEVERIFY is used when the program sends output to the window that depends on a knowledge of the current size of the window. If the user wants to resize the window, you may want to make sure that any queued output completes before the sizing takes place (critical text, for instance). To do so, set this flag. Then, when the user wants to size, Intuition will send the program the SIZEVERIFY message and Wait() until the program replies that it is all right to proceed with the sizing.

  NOTE: Saying that Intuition will Wait() until your program replies is really saying that the user will wait until the program replies, which suffers the great negative potential of user-unfriendliness. Remember to use this flag sparingly, and, as always with any IDCMP message your program receives, reply promptly! After the user has sized the window, your program can find out about it by using NEWSIZE.
- NEWSIZE is the flag that tells Intuition to send an IDCMP message after the user has resized your window. At this point, you could examine the size variables in your Window structure to discover the new size of the window.
- REFRESHWINDOW, when set, will cause a message to be sent whenever your window needs refreshing. This flag makes sense only with SIMPLEJtEFRESH and SMARTJtEFRESH windows.
- MOUSEBUTTONS will make sure your program receives reports about mouse-button up/down events. NOTE: Only the events that mean nothing to Intuition are reported. If the user clicks the select button over a gadget, Intuition deals with it without sending any message.
- MOUSEMOVE works only if you set the REPORTMOUSE flag (see above) or if one of your gadgets has the flag FOLLOWMOUSE set.

Then all mouse movements will be reported through the IDCMP.

- o GADGETDOWN specifies that when the user "selects" a gadget you have created with the GADGIMMEDIATE flag set, the fact will be broadcast through the IDCMP.
- o GADGETUP specifies that when the user "releases" a gadget that you have created with the RELVERIFY flag set, the fact will be broadcast through the IDCMP.
- o MENUPICK specifies that **MenuNumber** data be sent to your program.
- o CLOSEWINDOW specifies that the CLOSEWINDOW event be broadcasted through the IDCMP rather than the console device.
- o RAWKEY specifies that all RAWKEY events be transmitted via the IDCMP. Note that these are absolutely raw keycodes, which you will have to massage before using. Setting this and the MOUSE flags effectively eliminates the need to open **a** console device to get input from the keyboard and mouse. Of course, in exchange you lose all of the console features, most notably the "cooking" of input data and the systematic output of text to your window.
- o VANILLAKEY is the raw keycode RAWKEY event translated into the current default character keymap of the console device. In the USA, the default keymap is ASCII characters. When you set this flag, you will get IntuiMessages where the **Code** field has a character representing the key struck on the keyboard.
- o INTUITICKS gives you simple timer events from Intuition when your window is the active one; it may help you avoid opening and managing the timer device. With this flag set, you will get only one queued-up INTUITICKS message at a time. If Intuition notices that you've been sent an INTUITICKS message and haven't replied to it, another message will *not* be sent.

  Intuition receives timer events ten times a second (approximately).
- o Set ACTIVEWINDOW and INACTIVEWINDOW to discover when your window becomes activated or inactivated.

Gadgets = a pointer to the first of a linked list of your own gadgets that you want attached to this window. Can be NULL if you have no gadgets of your own.

CheckMark = a pointer to an instance of the Image structure that contains the imagery you want used when any of your **Menultems** is to be check-marked. If you do not want to supply your own imagery and prefer to use Intuitipn's own checkmark, set this argument to NULL.

Text = **a** null-terminated line of text that will appear on the title bar of your window (may be NULL if you want no text).

Type = the screen type for this window. If this equals CUSTOMSCREEN, you must have already opened a custom screen (see text above). Types available include:

- o WBENCHSCREEN
- o CUSTOMSCREEN

Screen = if your type is one of Intuition's standard screens, this argument is ignored. However, if type === CUSTOMSCREEN, this must point to the structure of your own screen.

BitMap == if you have specified SUPERJ3ITMAP as the type of raster you want for this window, this value points to a instance of the BitMap structure. However, if the raster type is not SUPERJBITMAP, this pointer is ignored.

MinWidth, MinHeight, MaxWidth, MaxHeight = the size limits for this window. These must be reasonable values, which is to say that the minimums cannot be greater than the current size, nor can the maximums -be smaller than the current size. If they are, they are ignored. Any one of these can be initialized to zero, which means that that limit will be set to the current dimension of that axis. The limits can be changed after the window is opened by calling the **WindowLimitsQ** routine. If you have not requested the WINDOWSIZING option, these variables are ignored and you do not have to initialize them.

RESULT

If all is well, this command returns a pointer to the structure of your new window. If anything goes wrong, it returns NULL.

BUGS

ACTIVATE is currently advisory only. The user is able to do things that will prevent your window from becoming the active one when it opens.

SEE ALSO

OpenScreenQ.
ModifyIDCMP().
SetWindowTitles().
WindowLimitsQ.

NAME

OpenWorkBench — Opens the Workbench screen.

SYNOPSIS

BOOL OpenWorkBench();

FUNCTION

This routine attempts to reopen the Workbench. If the Workbench screen reopens successfully, this routine returns TRUE; if something goes wrong, it returns FALSE.

Even though this routine does return a BOOL value, you can ignore the return value if you want.

INPUTS

None.

RESULT

TRUE if the Workbench screen opened successfully or was already opened.
FALSE if anything went wrong and the Workbench screen is not open.

BUGS

None.

SEE ALSO

None.

NAME

PrintlText — Prints the text according to the IntuiText argument.

SYNOPSIS

```
PrintIText(RastPort, IText, LeftEdge, TopEdge);
           AO        Al     DO        Dl
```

FUNCTION

This routine prints the **IntuiText** into the specified **RastPort. It** sets up the **RastPort** as specified by the **IntuiText** values, then prints the text into the **RastPort** at the **IntuiText** x,y coordinates offset by the left/top arguments.

This routine does Intuition window-clipping as appropriate. If you print text outside of your window, your characters will be clipped at the window's edge.

If the **NextText** field of the **IntuiText** argument is non-zero, the next **IntuiText** is drawn as well (return to the top of this **FUNCTION** section for details).

INPUTS

RastPort = pointer to the **RastPort** destination of the text.
IText = pointer to an **IntuiText** structure.
LeftEdge = left offset of the **IntuiText** into the **RastPort.**
TopEdge = top offset of the **IntuiText** into the **RastPort.**

RESULT

None.

BUGS

None.

SEE ALSO

None.

NAME
RefreshGadgets - Refreshes (redraws) the gadget display.

SYNOPSIS
```
RefreshGadgets(Gadgets, Pointer, Requester);
               AO       Al       A2
```

FUNCTION
This routine refreshes (redraws) all of the gadgets in the gadget list, starting from the specified gadget.

The **Pointer** argument points to a **Window** structure.

The **Requester** variable can point to a **Requester** structure. If the first gadget in the list has the REQGADGET flag set, the gadget list refers to gadgets in **a** requester and **Pointer** must necessarily point to a window. If these are not the gadgets of a requester, the **Requester** argument may be NULL.

There are two main reasons why you might want to use this routine. First, you have modified the imagery of the gadgets in your display and you want the new imagery to be displayed. Second, if you think that some graphic operation you just performed trashed the gadgetry of your display, this routine will refresh the imagery.

The **Gadgets** argument can be a copy of the FirstGadget variable in either the **Screen** or **Window** structure that you want refreshed; the effect of this will be that all gadgets will be redrawn. However, you can selectively refresh just some of the gadgets by starting the refresh part way into the list—for instance, redrawing your window non-GIMMEZEROZERO gadgets only, which you have conveniently grouped at the end of your gadget list.

NOTE: It is never safe to tinker with the gadget list yourself. Do not supply some gadget list that Intuition has not already processed in the usual way.

NOTE: If you have specified that this is the gadget list **of a** requester, that requester must be currently displayed.

INPUTS
Gadgets = pointer **to** the first structure in the list of gadgets wanting refreshment.
Pointer = pointer to a **Window** structure.
Requester = pointer to **a Requester** structure (may be NULL if this is not a requester gadget list).

RESULT

None.

BUGS

None.

SFE ALSO

None.

NAME

RemakeDisplay — Remakes the entire Intuition display.

SYNOPSIS

RemakeD isplay ();

FUNCTION

This is the big one. This procedure remakes the entire Intuition display. It calls **MakeScreen()** for every screen in the system and then it calls **RethinkDisplay**(), which rethinks the relationships of the screens to one another and then rethinks the display Copper lists.

WARNING: This routine can take several milliseconds to run, so do not use it lightly. **RethinkDisplayO** (called by this routine) does a **Forbid()** on entry and a **PermitQ** on exit, which can seriously degrade the performance of the multitasking Executive.

INPUTS

None.

RESULT

None.

BUGS

None.

SEE ALSO

RethinkDisplayQ.

NAME

RemoveGadget — Removes a gadget from a window.

SYNOPSIS

```
USHORT RemoveGadget(Pointer, Gadget);
                    AO       Al
```

FUNCTION

This routine removes the given gadget from the gadget list of the specified window. It returns the ordinal position of the removed gadget. If the gadget pointer points to a gadget that is not in the appropriate list, -1 is returned. If there are no gadgets in the list, -1 is returned. If you remove the 65,535th gadget from the list, -1 is returned.

NOTE: The gadget's imagery is *not* erased by this routine.

INPUTS

Pointer = pointer to the window from which the gadget is to be removed.

Gadget = pointer to the gadget to be removed. The gadget itself describes whether this gadget should be removed from the window.

RESULT

Returns the ordinal position of the removed gadget. If the gadget was not found in the appropriate list or if there are no gadgets in the list, -1 is returned.

BUGS

None.

SEE ALSO

AddGadgetQ.

NAME

ReportMouse — Tells Intuition whether or not to report mouse movement.

SYNOPSIS

```
ReportMousefWindow, Boolean);
                    AO       DO
```

FUNCTION

This routine tells Intuition whether or not to broadcast mouse movement events to this window when it is active. The Boolean value specifies whether to start or stop broadcasting position information of mouse-movement. If the window is active, mouse-movement reports start coming immediately after this command. This routine will change the current state of the FOLLOWMOUSE function of a currently-selected gadget, too. Note that calling **ReportMouse()** when a gadget is selected will only temporarily change whether or not mouse movements are reported while the gadget is selected; the next time the gadget is selected, its FOLLOWMOUSE flag is examined anew. Note also that calling ReportMouseQ when no gadget is currently selected will change the state of the window's REPORTMOUSE flag but will have no effect on any gadget that may be subsequently selected.

The ReportMouseQ function is first performed when **OpenWindowQ** is first called. If the flag REPORTMOUSE is included among the options, all mouse-movement events are reported to the opening task and will continue to be reported until ReportMouseQ is called with a Boolean value of FALSE. If REPORTMOUSE is not set, no mouse-movement reports will be broadcast until **ReportMouseQ** is called with a Boolean value of TRUE.

INPUTS

Window = pointer to a Window structure associated with this request.
Boolean = TRUE or FALSE value specifying whether to turn this function on or off.

RESULT

None.

BUGS

None.

SEE ALSO

None.

NAME

Request — Activates a requester.

SYNOPSIS

```
Request(Requester, Window);
        AO         Al
```

FUNCTION

This routine links in and displays a requester in the specified window. This routine ignores the window's REQVERIFY flag.

INPUTS

Requester = pointer to the structure of the requester to be displayed.
Window = pointer to the structure of the window into which this requester goes.

RESULT

If the requester is successfully opened, TRUE is returned. If the requester could not be opened, FALSE is returned.

BUGS

None.

SEE ALSO

None.

NAME

RethinkDisplay — The grand manipulator of the entire Intuition display.

SYNOPSIS

Ret hin kD isp lay ();

FUNCTION

This function performs the Intuition global display reconstruction. This includes massaging internal-state data, rethinking all of the **ViewPorts** and their relationship to one another, and, finally, reconstructing the entire display based on the results of all this rethinking.

The reconstruction of the display includes calls to the graphics library to perform **MrgCopQ** and **LoadViewQ** for all of Intuition's screens.

You may perform a **MakeScreenQ** on your custom screen before calling this routine. The results will be incorporated in the new display.

WARNING: This routine can take several milliseconds to **run,** so **do** not use it **lightly. RethinkDisplay()** does **a ForbidQ** on entry and a **Permit()** on exit, which can seriously degrade the performance of the multitasking Executive.

INPUTS

None.

RESULT

None.

BUGS

None.

SEE ALSO

MakeScreen().
RemakeDisplay().
The graphics library's **MrgCopQ** and **LoadViewQ.**
Exec's **Forbid()** and **PermitQ.**

NAME
ScreenToBack — Sends the specified screen to the back of the display.

SYNOPSIS
ScreenToBack(Screen);
A0

FUNCTION
This routine sends the specified screen to the back of the display.

INPUTS
Screen = pointer to a **Screen** structure.

RESULT
None.

BUGS
None.

SEE ALSO
ScreenToFront().

NAME

ScreenToFront — Brings the specified screen to the front of the display.

SYNOPSIS

```
ScreenToFront(Screen);
              AO
```

FUNCTION

This routine brings the specified screen to the front of the display.

INPUTS

Screen = a pointer to **a Screen** structure.

RESULT

None.

BUGS

None.

SEE ALSO

ScreenToBackQ.

NAME

SetDMRequest - Sets the DMRequest of the window.

SYNOPSIS

```
SetDMRequest(Window, DMRequester);
                A0       A1
```

FUNCTION

This routine attempts to set the DMRequester in the specified window. The DMRequester is the special requester that you attach to the double-click of the menu button, allowing the user to bring up this requester on demand. This routine will not set the DMRequester if it is already set and is currently active (in use by the user). To change the DMRequester after having called **SetDMRequestQ,** you start by calling ClearDMRequest() until it returns a value of TRUE. Then you can call **SetDMRequestQ** with the new DMRequester.

INPUTS

Window = pointer to the structure of the window into which the DMRequest is to be set.
DMRequester = a pointer to a **Requester** structure.

RESULT

If the current DMRequest was not in use, the **DMRequester** pointer is set in the window and this routine returns TRUE.

If the DMRequest was currently in use, this routine does not change the pointer and returns FALSE.

BUGS

None.

SEE ALSO

ClearDMRequestQ.
Request Q.

NAME

SetMenuStrip — Attaches the menu strip to the window.

SYNOPSIS

```
SetMenuStrip(Window, Menu);
             AO      Al
```

FUNCTION

This routine attaches the menu strip to the window. If the user presses the menu button after this routine is called, this specified menu strip will be displayed and accessible.

NOTE: You should always design your menu strip changes to be two-way operations; every menu strip you add to your window should be cleared sometime. Even in the simplest case, when you will have just one menu strip for the lifetime of your window, you should always clear the menu strip before closing the window. If you already have a menu strip attached to this window, the correct procedure for changing to a new menu strip involves calling ClearMenuStripQ to clear the old menu strip first. The sequence of events should be:

1. **OpenWindowQ.**
2. Zero or more iterations of:
   - **SetMenuStrip().**
   - **ClearMenuStripQ.**
3. **CloseWindow().**

INPUTS

"**Window** = pointer to a **Window** structure.
Menu = pointer to the first **Menu** structure in the menu strip.

RESULT

None.

BUGS

None.

SEE ALSO

ClearMenu Strip ().

NAME
SetPointer — Sets a window with its own pointer.

SYNOPSIS
```
SetPointer(Window, Pointer, Height, Width, XOffset, YOffset);
           AO      Al       DO      Dl     D2       D3
```

FUNCTION
This routine sets up the window with the sprite definition for the pointer. Then, whenever the window is active, the pointer image will change to the sprite's version of the pointer. If the window is active when this routine is called, the change takes place immediately.

The **XOffset** and **YOffset** arguments are used to offset the top left corner of the hardware sprite imagery from what Intuition regards as the current position of the pointer. Another way of describing it is as the offset from the "hot spot" of the pointer to the top left corner of the sprite. For instance, if you specify offsets of zero, zero, then the top-left corner of your sprite image will be placed at the pointer position. On the other hand, if you specify an **XOffset** of **-7** (remember, sprites are 16 pixels wide), your sprite will be centered over the pointer position. If you specify an **XOffset** of **-15,** the right edge of the sprite will be over the pointer position.

INPUTS
Window = pointer to the structure of the window to receive this pointer definition.
Pointer = pointer to the data definition of **a** sprite.
Height = the height of the pointer.
Width = the width of the sprite (must be less than or equal to 16).
XOffset = the offset for your sprite from the pointer position.
YOffset = the offset for your sprite from the pointer position.

RESULT
None.

BUGS
None.

SEE ALSO
ClearPointerQ.

NAME

SetWindowTitles -- Sets the window's titles for both the window and the screen.

SYNOPSIS

```
SetWindowTitles(Window, WindowTitle, ScreenTitle);
                A0      A1           A2
```

FUNCTION

This routine allows you to set the text that appears in the window and/or screen title bars. The window title appears at all times in the window title bar. The window's screen title appears at the screen title bar whenever this window is active.

When this routine is called, your window title will be changed immediately. If your window is active when this routine is called, the screen title will be changed immediately.

You can specify a value of -1 (negative one) for either of the title pointers. This designates that you want Intuition to leave the current setting of that particular title alone, modifying only the other one. Of course, you could set both to -1.

Furthermore, you can set a value of 0 for either of the title pointers. Doing so specifies that you want no title to appear (the title bar will be blank).

INPUTS

Window = pointer to your **Window** structure.
WindowTitle = pointer to a null-terminated text string; this pointer can also be set to either -1 (negative one) or 0 (zero).
ScreenTitle = pointer to a null-terminated text string; this pointer can also be set to either -1 (negative one) or 0 (zero).

RESULT

None.

BUGS

None.

SEE ALSO

OpenWindow()-
ShowTitleQ-

NAME

ShowTitle — Sets the screen title bar display mode.

SYNOPSIS

```
ShowTitle(Screen, Showlt);
           AO      DO
```

FUNCTION

This routine sets the SHOWTITLE flag of the specified screen and then coordinates the redisplay of the screen and its windows.

The screen title bar can appear either in front of or behind Backdrop windows. Non-Backdrop windows always appear in front of the screen title bar. You specify whether you want the screen title bar to be in front of or behind the screen's Backdrop windows by calling this routine.

The **Showlt** argument should be set to either TRUE or FALSE. If TRUE, the screen's title bar will be shown in front of Backdrop windows. If FALSE, the title bar will be located behind all windows. When a screen is first opened, the default setting of the SHOWTITLE flag is TRUE.

INPUTS

Screen = pointer to a Screen structure.
Showlt = Boolean TRUE or FALSE describing whether to show or hide the screen title bar.

RESULT

None.

BUGS

None.

SEE ALSO

SetWindowTitlesQ.

NAME

SizeWindow — Asks Intuition to size a window.

SYNOPSIS

```
SizeWindow(Window, DeltaX, DeltaY);
           AO      DO      Dl
```

FUNCTION

This routine sends a request to Intuition asking to size the window by the specified amounts. The delta arguments describe how much to size the window along the respective axes.

Note that the window will not be sized immediately. It will be sized the next time Intuition receives an input event, which happens currently at **a** minimum rate of ten times per second and a maximum of sixty times a second. You can discover when your window has finally been sized by setting the NEWSIZE flag of the IDCMP of your window. See the "Input and Output Methods" chapter of this book (chapter 8) for a description of the IDCMP.

This routine does no error-checking. If your delta values specify some far corner of the universe, Intuition will attempt to size your window to the far corners of the universe. Because of the distortions in the space-time continuum that can result from this, as predicted by special relativity, the result is generally not a pretty sight.

INPUTS

"Window = pointer to the structure of the window to be sized.
DeltaX = signed value describing how much to size the window on the x axis.
DeltaY = signed value describing how much to size the window on the y axis.

RESULT

None.

BUGS

None.

SEE ALSO

MoveWindow().
WindowToFront().
WindowToBackQ.

NAME

ViewAddress - Returns the address of the Intuition View structure.

SYNOPSIS

View Ad dressQ;

FUNCTION

This routine returns the address of the Intuition **View** structure. If you want to use any of the graphics, text, or animation primitives in your window and that primitive requires a pointer to a **View,** this routine will return the address of the **View** for you.

INPUTS

None.

RESULT

Returns the address of the Intuition **View** structure.

BUGS

It would be hard **for** this routine to have a bug.

SEE ALSO

All of the graphics, text, and animation primitives.

NAME

ViewPortAddress - Returns the address of a window's ViewPort structure.

SYNOPSIS

ViewPortAddress(Window);
A0

FUNCTION

This routine returns the address of the **ViewPort** structure associated with the specified window. This is actually the **ViewPort of** the screen within which the window is displayed. If you want to use any of the graphics, text, or animation primitives in your window and that primitive requires a pointer to a **ViewPort** structure, you can use this call.

INPUTS

Window = pointer to the **Window** structure for which you want the **ViewPort** address.

RESULT

Returns the address of the window's **ViewPort** structure.

BUGS

It would be **hard for** this routine to have a bug.

SEE **ALSO**

All of the graphics, text, and animation primitives.

NAME

WBenchToBack — Sends the Workbench screen in back of all screens.

SYNOPSIS

WBenchToBackQ;

FUNCTION

This routine causes the Workbench screen, if it is currently opened, to go to the background. This does not "move" the screen up or down; it affects only the depth arrangement of the screen.

If the Workbench screen was opened, this function returns TRUE; otherwise, it returns FALSE.

INPUTS

None.

RESULT

If the Workbench screen was opened, this function returns TRUE; otherwise, it returns FALSE.

BUGS

None.

SEE ALSO

WBenchToFrontQ.

NAME

WBenchToFront — Brings the Workbench screen in front of all screens.

SYNOPSIS

WBenchToFront();

FUNCTION

This routine causes the Workbench screen, if it is currently opened, to come to the foreground. This does not "move" the screen up or down; it affects only the depth arrangement of the screen.

If the Workbench screen was opened, this function returns TRUE; otherwise, it returns FALSE.

INPUTS

None.

RESULT

If the Workbench screen was opened, this function returns TRUE; otherwise, it returns FALSE.

BUGS

None.

SEE ALSO

WBenchToBackQ.

NAME

WindowLimits — Sets the minimum and maximum limits of the window.

SYNOPSIS

```
WindowLimits(Window, MinWidth, MinHeight, MaxWidth, MaxHeight);
             AO      DO        Dl         D2        D3
```

FUNCTION

This routine allows you to adjust the minimum and maximum limits of the window's size. Until this routine is called, the window's size limits are equal to the initial limits specified by the call to **OpenWindowQ.**

If you do not want to change any one of the dimensions, set the limit argument for that dimension to zero. If any limit argument is equal to zero, that argument is ignored and the initial setting of that parameter remains undisturbed.

If any argument is out of range (minimums greater than the current size, maximums less than the current size), that limit will be ignored, though the others will still take effect if they are in range. If any argument is out of range, the return value from this procedure will be FALSE. If all arguments are valid, the return value will be TRUE.

If the user is currently sizing this window, the new limits will not take effect until after the sizing is completed-

INPUTS

Window = pointer to a Window structure.

MinWidth, MinHeight, **MaxWidth, MaxHeight** = the new limits for the size of this window. If a limit is set to zero, it will be ignored and that setting will be unchanged.

RESULT

Returns TRUE if everything was in order. If a parameter was out of range (minimums greater than current size, maximums less than current size), FALSE is returned, and the errant limit request is not fulfilled (though the valid ones will be).

BUGS

None.

SEE ALSO

OpenWindowQ

NAME

WindowToBack — Asks Intuition to send this window to the back.

SYNOPSIS

WindowToBack(Window)j
AO

FUNCTION

This routine sends a request to Intuition asking to send the window in back of all other windows in the screen. Note that the window will not be depth arranged immediately; it will be arranged the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second and a maximum of sixty times a second.

Remember that Backdrop windows cannot be depth-arranged.

INPUTS

Window = pointer to the structure of the window to be sent to the back.

RESULT

None.

BUGS

None.

SEE ALSO

Mo veWiudow ().
SizeWindow().
WindowToFrontQ.

NAME

WindowToFront -- Asks Intuition to bring this window to the front.

SYNOPSIS

WindowToFront(Window);
AO

FUNCTION

This routine sends a request to Intuition asking to bring the window in front of all other windows in the screen.

Note that the window will not be depth-arranged immediately. It will be arranged the next time Intuition receives an input event, which happens currently at a minimum rate of ten times per second and a maximum of sixty times **a** second.

Remember that Backdrop windows cannot be depth arranged.

INPUTS

Window = pointer to the structure of the window to be brought to front.

RESULT

None.

BUGS

None.

SEE ALSO

MoveWindow ().
SizeWindowQ.
WindowToBackQ.

Appendix C

INTERNAL PROCEDURES

This appendix discusses the more esoteric and internal Intuition functions. These functions are definitely *not* for the casual user. Using these functions can seriously alter the user's environment, which is potentially a hazardous thing to do. You have more leeway when using these functions in a machine environment in which you have taken complete control of the Amiga and do not intend to allow other tasks to coexist with yours. However, if you intend to have your program run in the multitasking environment, please use these routines thoughtfully. The effects on other people's programs and on the user's understanding of the normal course of events can be dramatic at best, and can cause serious loss of data and loss of the user's confidence in using the Amiga.

With that caveat aside, here are the functions covered in this appendix:

SetPrefs()
This routine allows you to set Intuition's internal state of the Preferences.

AlohaWbrkbenchQ
This routine allows the Workbench tool to make its presence and departure known to Intuition.

Intuition()
This is the main entry point into Intuition, where input events arrive and are dispatched.

SetPrefsQ

This routine configures Intuition's internal data states according to the specified Preferences structure. Normally, this routine is called only by:

- o The Preferences program itself after the user has changed the Preferences. The Preferences program also saves the user's Preferences data into a disk file named devs:system-configuration.

- o AmigaDOS when the system is being booted up. AmigaDOS opens the devs:system-configuration file and passes the information found there to the **SetPrefs()** routine. This way, the user can create an environment and have that environment restored every time the system is booted.

Note that the intended use for the **SetPrefsQ** call is entirely to serve the user. You should never use this routine to make your programming or design job easier at the cost of yanking the rug out from beneath the user.

The synopsis of this function is:

SetPrefs(Preferences, Size, RealThing)

Preferences - a pointer to a Preferences structure

Size - the number of bytes contained in your Preferences structure. Typ^ ally, you will use "sizeof(struct Preferences)" for this argument.

RealThing - a Boolean TRUE or FALSE designating whether or not this is an intermediate or final version of the Preferences. The difference is that final changes to Intuition's preferences causes a global broadcast of NEWPREFS events to every application that is listening for this event. Intermediate changes may be used, for instance, to update the screen colors while the user is playing with the color gadgets.

Refer to chapter 11, "Other Features," for information about the Preferences structure and the standard Preferences procedure calls.

AlohaWorkbenchQ

In Hawaiian, "aloha" means both hello and goodbye. The **AlohaWorkbench()** routine allows the Workbench program to inform Intuition that it has become active and that it is shutting down.

If the Workbench program is active, Intuition is able to tell it to open and close its windows when someone uses the Intuition OpenWorkBenchQ and **CIoseWorkBench()** functions to open or close the Workbench screen. If the Workbench program is not active, presumably it has no opened windows, so there is no need for this communication.

This routine is called with one of two kinds of arguments—either a pointer to an initialized message port (which designates that Workbench is active and communications can take place), or NULL to designate that the Workbench tool is shutting down.

When the message port is active, Intuition will send IntuiMessages to it. The messages will have the Class field set to WBENCHMESSAGE. The **Code** field will equal either WBENCHOPEN or WBENCHCLOSE, depending on whether the Workbench application should open or close its windows. Intuition assumes that Workbench will comply, so as soon as the message is replied to, Intuition proceeds with the expectation that the windows have been opened or closed accordingly.

The procedure synopsis is:

AlohaWorkbenchfWBPort)

WBPort - a pointer to an initialized **MsgPort** structure in which the special communications are to take place.

IntuitionQ

This is Intuition's main entry point. All of Intuition's I/O operations originate here. The input stream flows into Intuition at this portal.

This routine accepts a single argument: a pointer to a linked list of **InputEvent** structures. These events have all the real-time state information that Intuition needs to create its art. Refer to the *Amiga ROM Kernel Manual* for more information about **InputEvent** structure and the operation of the input device.

When **IntuitionQ** exits, it returns a pointer to a linked list of **InputEvent** structures. This list of **InputEvents** has no dependable correspondence to the list that was initially submitted to **IntuitionQ.** Intuition may add events to the list and extract events from the list. This list of events is normally intended for the console device.

If you are considering feeding false input events to Intuition, please think again. If you are running in an environment in which you have taken over the machine, it is probably safe to fool Intuition in a controlled way. If you are running in a multitasking environment, however, especially one in which the input device is still feeding input events directly into the stream, you can easily cause more harm than good. You may not be able to anticipate the things that could go wrong when other programs try to exist in an environment that you are modifying.

If you are determined to feed false input events to Intuition, it is much safer to add an input handler to the system than to call Intuition(). Add the input handler to the system at a priority higher than Intuition's; the input queue priority of Intuition is 50, so **a** priority of 51 will suffice. This will allow your program to see *all* input events before Intuition sees them. You can filter the input events, allowing Intuition to see only those events that you want it to see. Also, you can add synthesized events to the input event stream. This allows you to fool Intuition in an honest, system-integrated way.

For example, say that you want to position the pointer yourself but you want to let the user interact with the rest of the system as usual. If you see mouse movement events, you can filter them out and not let Intuition see them. At the same time, you can create mouse movement events of your own. On the other hand, if you see keyboard events you can leave them undisturbed. See the *Amiga ROM Kernel Manual* for details about the input-handler queue.

An important note is that **IntuitionQ** is sometimes required to call the Exec **WaitQ** function. Normally, **IntuitionQ** is called from within the input device's task, so the input device enters the wait state when these situations arise. If you call **IntuitionQ** directly, **your** task may h#ve to wait. The obvious problem here is the classic lockout problem—your task cannot create the required response because **your** task has forced itself to wait, which will cause the system to freeze. The best way to get around thb is to have a separate task that calls **IntuitionQ** and does nothing more.

The synopsis of this function is:

Intu ition (InputEvent)

InputEvent - a pointer to the first in a linked list of **InputEvent** structures.

itself to wait, which will cause the system to freeze. The best way to get around ***ih.k*** is to have a separate task that calb IntuitionQ and does nothing more.

The synopsis of this function is:

Intuition(InputEvent)

InputEvent - a pointer to the first in a linked list of **InputEvent** structures.

GLOSSARY

Active screen	The screen containing the active window.
Active window	The window receiving user input. Only one window is active at a time.
Alert	Information exchange device displayed by the system or the application when serious problems occur or when immediate action is necessary.
ALT keys	Two command keys on the keyboard to the left and right of the Amiga keys.
Alternate	An image or border used in gadget highlighting. When the gadget is selected, the alternate image or border is substituted for the original image or border.
Amiga keys	Two command keys on the keyboard to the left and right of the space bar.
AmigaDOS	The Amiga disk operating system.
Application gadget	A custom gadget created by the developer.
Auto-knob	The special automatic knob for proportional gadgets; changes its shape according to the current proportional settings.
Backdrop wincjow	A window that stays anchored to the back of the display.
Bit-map	The complete definition of a display in memory, consisting of one or more bit-plaises and information about how to organize the rectangular display.
Bit-plane	A contiguous series of memory words, treated as if it were a rectangular shape.

Body variables	**Proportional gadget variables that contain the increment by which the pot variables may change.**
Boolean gadget	**A simple yes-or-no gadget.**
Border area	**The area containing border gadgets.**
Border line	**The default double-line drawn around the perimeter of all windows, except Borderless windows.**
Borderless window	**A window with no drawn border lines.**
Buffer	**An area of continuous memory, typically used for storing blocks of data such as text strings.**
Checkmark	**A small image that appears next to a menu item showing that the user has selected that item. By default, the checkmark is** *y/* **, but a custom image can be substituted.**
CLI	***See* Command Line Interface.**
Click	**To quickly press and release a mouse button.**
Clipboard	**A Workbench file used to store the last data cut (removed) from a project.**
Clipping	**Causing a graphical rendering to appear only in some bounded area, such as only within the non-concealed areas of a window.**
Close	**To remove a window or screen from the display.**
Close gadget	**Gadget in the upper left corner of a screen or window that the user selects to request that a window or screen be closed.**
Color indirection	**The method used by Amiga for coloring individual pixels, in which the binary number formed from all the bits that define a given pixel refers to one of the 32 color registers. Each of the 32 color registers can be set equal to any of 4,096 colors.**
Color palette	**The set of colors available in a screen.**

Color register	One of 32 hardware regbters containing colors that you can define.
Column	A set of adjoining pixels that forms a vertical line on the video display.
Command keys	Keys that combine with alphanumeric keys to create command key sequences, which substitute for making selections with the mouse buttons.
Command Line Interface	The conventional interface to system commands and utilities.
Complement	The binary complement of a color, used as a method of gadget highlighting and in flashing the screen. To complement a binary number means to change all the Is to Os and all the Os to Is.
Console devibe	A communication path for both user input and program output. Especially recommended for input/output of text-only applications.
Container	Part of a proportional gadget; the area within which the knob or slider can move; the select box of the gadget.
Control escape sequence	Special sequences of characters that start with the "Escape" character.
Controller	A hardware device, such as a mouse or a light pen, used to move the pointer or furnish some other input.
Coordinates	A pair of numbers shown in the form (x,y), where x is an offset from the left side of the display or display component and y is an offset from the top.
Copper	Display-synchronized coprocessor that handles the Amiga video display.
Coprocessor	*See* Copper.
Cursor keys	The arrow keys, which can be used as a substitute for using the mouse to move the pointer.
Data structure	The grouping together of the components required to define some data element.

Depth	Number of bit-planes in a display.
Depth-arrangement gadgets	Gadgets in the title bar of a screen or window used to send the screen or window to the back of the display or bring it up front.
Disable	To make something unavailable to the user.
Display	To put up a screen, window, requester, alert, or any other graphics object on the video display.
Display field	One complete scanning of the video beam from top to bottom of the video display screen.
Display memory	The RAM that contains the information for the display imagery; the hardware translates the contents of the display memory into video signals.
Display modes	Display parameters set in the definition of a screen. The modes are high or low horizontal resolution, interlaced or non-interlaced vertical resolution, sprite mode, and dual-playfield mode.
Double-click	To quickly press and release a mouse button twice.
Double-menu requester	A requester that the user can open by double-clicking the mouse menu button.
Drag	To move an icon, gadget, window, or screen by placing the pointer over the object to be moved and holding down the selection button while moving the mouse.
Drag gadget	The portion of a window or screen title bar that contains no other gadgets, used for moving a window or screen around on the video display.
Dual-playfield mode	A display mode that allows you to manage two separate display memories, giving you two separately controllable displays at the same time.
Edit menu	A menu for text processing that includes various text-editing functions.
Enable	To make something available to the user; a menu item or gadget that is enabled £an be selected by the user.

Exec	Low-level primitives that comprise the Amiga multitasking operating system.
Extended selection	A technique for selecting more than one menu item at a time.
Fill	To put a color or pattern within an enclosed area.
Flag	A mechanism for selecting an option or detecting a state; a name representing a bit to be set or cleared.
Font	A set of letters, numbers, and symbols that share the same basic design.
Gadget	Any of the control devices provided within a window, screen, or requester; employed by users to change what is being displayed or to communicate with an application or with Intuition.
Ghost	Display less distinctly (overlay an area with a faint pattern of dots) to indicate that something, such as a gadget or a window, is not available or not active.
Ghost shape	The new outline of a window that shows briefiy when the user is dragging or sizing a window.
Gimmezerozero window	A window with a separate bit-map for the window border.
Header file	A file that is included at the beginning of a C program and contains definitions of data types and structures, constants, and macros.
High-resolution mode	A horizontal display mode in which 640 pixels are displayed across a horizontal line.
Highlight	To modify the display of a selected menu item or gadget in a way that distinguishes it from its non-selected state.
Hit select	A method of gadget selection in which the gadget is unselected as soon as the select button is released.
Hold-and-modify mode	A display mode that gives you extended color selection — up to 4,096 colors on the screen at one time.
Hue	The characteristic of a color that is determined by the color's position in the color spectrum.

Icon	A visual representation of an object in the Workbench, such as a program, file, or disk.
IDCMP	"Intuition Direct Communications Message Ports"; the primary communication path for user input to an application. Gives mouse and keyboard events and Intuition events in raw form. Provides a path for communicating to Intuition.
Initialize	To set up an Intuition component with certain default parameters.
Input event	The message created by the input device whenever a signal is detected at one of the Amiga input ports.
Interlaced mode	A vertical display mode in which 400 lines are displayed from top to bottom of the video display.
IntuiMessage	The input message created by Intuition for application programs; the message is the medium in this case.
KeyMap	Translation table used by the console device to translate keycodes into normal characters.
Knob	Part of a proportional gadget; the user manipulates the knob to set a proportional value.
Library	A collection of predefined functions that can be used by any program.
Linked list	A collection of like objects linked together by having a pointer variable in one contain the address of the next; the last object in the list has a next-pointer of NULL.
Low-resolution mode	A horizontal display mode in which 320 pixels are displayed across a horizontal line.
Menu	A category that has menu items associated with it. One of the entries in the menu list displayed in the screen title bar.
Menu bar	A strip in the screen title bar that shows the menu list when the user holds down the menu button.
Menu button	The right-hand button on the mouse.

Menu item	One of the choices in a menu; the options presented to the user.
Menu list	List of menus displayed in the screen title bar when the user holds down the menu button.
Menu shortcut	An alternate way of choosing a menu item by pressing a key on the keyboard while holding down the right AMIGA key.
Menu title	*See* Menu.
Message poris	A software mechanism managed by the Amiga Exec that allows intertask communications.
Mouse	A controller device used to move the pointer and make selections.
Multitasking	A system in which many tasks can be operating at the same time, with no task forced to be aware of any other task.
Mutual exclusion	The principle that says that selecting a menu item (or gadget) can cause other menu items (or gadgets) to become deselected.
Non-interlaced mode	A display mode in which 200 lines are displayed from top to bottom of the video display.
Null-terminated	A string that ends with a byte of zero; text strings must be null-terminated.
OS'set	A position in the display that is relative to some other position.
Open	For the user, to display a window. For an application, to display a window or screen.
Option	A feature that, once selected, persists until it is deselected.
Parallel port	A connector on the back of the Amiga used to attach printers and other add-ons.
Pen	A variable containing a color register number used for drawing lines or filling background.

Pixel	Short for "picture element." The smallest addressable element in the video display. Each pixel is one dot of color.
Playfield	One of the basic elements in Amiga graphics; the background for all the other display elements.
Pointer	A small object, usually an arrow, that moves on the display when the user moves the mouse (or the cursor keys). It is used to choose menu items, open windows, and drag and select other objects.
Pot variables	Proportional gadget variables that contain the actual proportional values.
Preferences	A program that allows the user to change various settings of an Amiga.
Preserve	To keep overlapped portions of the display in hidden memory buffers.
Primitives	Amiga low-level library functions.
Project menu	A menu for opening and saving project files.
Proportional gadget	A gadget used to display a proportional value or get a proportional setting from the user. Consists of a knob or slider and a container.
RAM	Random access (volatile) memory.
Raster	The area in memory where the bit-map is located.
RastPort	The data structure that defines the general parameters of display memory.
Refresh	To recreate a display that was hidden and is now revealed.
Render	To draw or write into display memory.
Requester	A rectangular information exchange region in a window. When a requester appears, the user must select a gadget in the requester to close the requester before doing anything else in the window.

Resolution	On a video display, the number of pixels that can be displayed in the horizontal and vertical directions.
Screen	A full-width area of the display with a set color palette, resolution, and other display modes. Windows open in screens.
Scroll	To move the contents of display memory within a window.
Scroll bar	A proportional gadget with which the user can display different parts of the display memory.
Select	To pick a gadget or menu item.
Select box	The sensitive area of a gadget or menu item. When the user moves the pointer within a gadget's select box, the gadget becomes selected.
Select button	The left-hand button on a mouse.
Selected option	An option that is currently in effect.
Selection shortcut	A quick way to select a gadget by pressing some key while holding down the left Amiga key.
Serial port	A connector on the back of the Amiga used to attach modems and other serial add-ons.
Shortcut	A quick way, from the keyboard, to choose a menu item or select a gadget.
Simple Refresh	A method of refreshing window display in which concealed areas are redrawn by the program when they are revealed
Size	To change the dimensions of a window or screen.
Sizing gadget	A gadget for the user to change the size of a window or a screen.
Slider	Part of a proportional gadget; used to pick a value within a range by dragging the slider or by moving the slider by increments with clicks of a mouse button.

Smart Refresh	A method of refreshing window display in which Intuition keeps information about concealed areas in off-display buffers and refreshes the display from this information. If the window is sized, the program may have to recreate the display.
Sprite	A small, easily movable graphic object. You can have multiple sprites in a window at the same time.
Sprite mode	A display mode that allows you to have sprites in your windows.
String gadget	A gadget that prompts the user to enter a text string or an integer.
Structure	*See* data structure.
Submenu	An additional menu that appears when some menu items are chosen by the user.
SuperBitMap Refresh	A method of window refresh where the display is recreated from a separate bit-map area.
SuperBitMap Window	A window with its own bit-map; doesn't use the screen's bit-map.
System gadgets	Predefined gadgets for windows and screens; for screens, dragging and depth arranging; for windows, dragging, depth arranging, sizing, and closing.
Task	Operating system module or application program. Each task appears to have full control over its own virtual 68000 machine.
Text cursor	In programs containing text and in string gadgets, a marker that indicates your position in the text.
Title bar	A strip at the top of a screen or window that contains gadgets and an optional name for the screen or window.
Toggle select	A method of gadget selection in which the gadget remains selected when the user releases the select button and does not become deselected until the user picks it again.
Tool	An application program.

Topaz	The default system font. It is a fixed-width font in two sizes: 60 columns wide and 8 lines tall; 80 columns wide and 9 lines tall.
Transparent	A special color register definition that allows a background color to show through. Used in dual-play field mode.
Type style	A variation of a typeface, such as italic or bold.
Typeface	See Font.
Undo	A text editing function that reverses an action.
UserPort	The message port created for you when you request IDCMP functionality. Your program receives messages from Intuition via this port.
Vector	A line segment.
Video display	Everything that appears on the screen of a video monitor or television.
View	The graphics library data structure used to create the Intuition display.
ViewPort	The graphics library data structure used to create and manage the Intuition screen.
Virtual terminal	An Intuition window; it accepts input from the user and displays output from the application.
Window	Rectangular display in a screen that accepts input from the user and displays output from the application.
WindowPort	The message port created for you when you request IDCMP functionality. You respond to messages from Intuition via this port.
Workbench	A program to manipulate AmigaDOS disk file objects.
Workbench screen	The primary Intuition screen.

Index

Appendix B

INTUITION INCLUDE FILE

This appendix contains the Intuition "include" file, which contains the definitions of all the Intuition data types and structures, constants, and macros. You include this file in all Intuition-based applications.

```
#ifndef  INTlflTIONJNTUITIONJH
#define INTUITIONJNTUITIONJI TRUE

/ *•* intuitjion.h **************************************************************
  *  Commodore-Amiga, Inc.
  *
  *  intuition.h main include for c programmers
  *
  *     Modification History
  *  date        : author:        Comments
  *  -------     --------         ---------------------------
  *  1-30-85     -=RJ=-           created this file!
  *  10-03-^5                     Support for HP printers
  *
  ******************************************************************************/

#ifhdef  INTUITIONJNTUITIONBASELH
^include  "intuition/intuitionbase.h"
#endif

#ifndef  GRAPHICS.GFXJi
#include  "gfaphics/gfx.h"
#endif

#ifndef  GRAPHICS.CLIPJH
#include  "gifaphics/clip.h"
#endif

#ifndef  GRAPHICS_VIEW_H
#include  "graphics/view.h"
#endif

#ifndef  GRAPHICS^RASTPORTJJ
#include  "graphics/rastport.h*
#endif

#ifndef  GRAPHICSJLAYERSJH
#include  "graphics/layers.h"
#endif

#ifndef  GRAPHICSLTEXTJH
#include  "graphics/text.h1*
#endif

#ifndefEXEl C:J>ORTSJI
^include  "exec/ports.h"
#endif

#ifndef  DEVICES.TIMERJi
#include  "devices/timer.h"
#endif
```

```
#ifndef DEVICES JNPUTEVENTJH
#include "devipes/inputevent.h"
#endif


/* ================================================================ */
/* === Border ===================================================== */
/* ================================================================ */
/* Data type Border, used for drawing a series of lines which is intended for
 * use as a border drawing, but which may, in fact, be used to render any
 * arbitrary vector shape.
 * The routine DrawBorder sets up the RastPort with the appropriate
 * variables, then does a Move to the first coordinate, then does Draws
 * to the subsequent coordinates.
 * After all the Draws are done, if NextBorder is non-zero we call DrawBorder
 * recursively.
 */
struct Border
{
  SHORT    L^ftEdge, TopEdge;      /* initial offsets from the origin */
  UBYTE    FriontPen, BackPen;     /* pen numbers for rendering */
  UBYTE    DrawMode;               /* mode for rendering */
  BYTE     Count;                  /* number of XY pairs */
  SHORT    *XY;                    /* vector coordinate pairs relative to LeftTop */
  struct   Border *NextBorder;     /* pointer to any other Border too */
};


/* ================================================================ */
/* [illegible] ==================================================== */
/* ================================================================ */
struct Gadget
{
  struct    Gadget *NextGadget;     /* next gadget in the list */

  SHORT    Le^Edge, TopEdge;       /* "hit box" of gadget */
  SHORT    Width, Height;          /* "hit box" of gadget */

  USHORT   Flfgs;                  /* see below for list of defines */

  USHORT   Aqtivation;             /* see below for list of defines •/

  USHORT   GadgetType;             /* see below for defines */

           /* appliprog can specify that the gadget be rendered as either as Border
            * or an Image.  This variable points to which (or equals NULL if there's
            * nothing to be rendered about this gadget)
            */
  APTR     GadgetRender;
```

```
        /* appliprog can specify "highlighted" imagery rather than algorithmic
         * this can point to either border or image data
         */
  APTR    SelectRender;

  struct  IntuiText *GadgetText;   /* text for this gadget */

        /* by using the MutualExclude word, the appliprog can describe
         * which gadgets mutually-exclude which other ones.  The bits
         * in MutualExclude correspond to the gadgets in object containing
         * the gadget list.  If this gadget is selected and a bit is set
         * in this gadget's MutualExclude and the gadget corresponding to
         * that bit is currently selected (e.g., bit 2 set and gadget 2
         * is currently selected) that gadget must be unselected.
         * Intuition does the visual unselecting (with checkmarks) and
         * leaves it up to the program to unselect internally
         */
  LONG    MutualExclude;           /* set bits mean this gadget excludes that gadget */

        /* pointer to a structure of special data required by proportional,
         * string and integer gadgets
         */
  APTR    SpecialInfo;

  USHORT  GadgetID;                /* user-definable ID field */
  APTR    UserData;                /* ptr to general-purpose user data (ignored by In) */
};

/* — FLAGS SET BY THE APPLIPROG———————————————————————*/
/* combinations in these bits describe the highlight technique to be used */
#define GADGHIGHBITS   0x0003
#define GADGHCOMP      0x0000    /* complement the select box */
#define GADGHBOX       0x0001    /* draw a box around the image */
#define GADGHIMAGE     0x0002    /* blast in this alternate image */
#define GADGHNONE      0x0003    /* don't highlight */

        /* set this flag if the GadgetRender and SelectRender point to Image imagery,
         * clear if it's a Border
         */
#define GADGIMAGE      0x0004

/*      * combinations in these next two bits specify to which corner the gadget's
        * Left & Top coordinates are relative.  If relative to Top/Left,
        * these are "normal" coordinates (everything is relative to something in
        * this universe)
        */
#define GRELBOTTOM     0x0008    /* set if rel to bottom, clear if rel top */
#define GRELRIGHT      0x0010    /* set if rel to right, clear if to left */
/* set the GRELWIDTH bit to spec that Width is relative to width of screen */
#define GRELWIDTH      0x0020
/* set the GRELHEIGHT bit to spec that Height is rel to height of screen */
```

```
#define GRELHEIGHT      0x0040

/* the SELECTEP flag is initialized by you and set by Intuition.  It
 * specifies whether or not this gadget is currently selected/highlighted
 */
#define SELECTED            0x0080

/* the GADGDISABLED flag is initialized by you and later set by Intuition
 * according to your calls to On/OffGadget().  It specifies whether or not •
 * this gadget is currently disabled from being selected
 V
#define GADGDISABLED        0x0100

/* — These are the Activation flag bits.................................*/
/* RELVERIFY is set if you want to verify that the pointer was still over
 *  the gadget when the select button was released
 */
#define RELVERIFY           0x0001

/* The flag GADGMMEDIATE, when set, informs the caller that the gadget
 * was activated when it was activated.  This flag works in conjunction with
 * the RELVERIFY flag
 V
#define GADGMMEDIATE        0x0002

/* The flag ENDGADGET, when set, tells the system that this gadget, when
 * selected, causes the requester or AbsMessage to be ended.  Requesters or
 * AbsMessages that are ended are erased and unlinked from the system
 */
#define ENDGAt>GET          0x0004

/* the FOLLOWMOUSE flag, when set, specifies that you want to receive
 * reports on mouse movements (i.e., you want the REPORTMOUSE function for
 * your window).  When the gadget is deselected (imi ^diately if you have
 * no RELVERIFY) the previous state of the REPORT MOUSE flag is restored
 * You probably want to set the GADGMMEDIATE uig when using FOLLOWMOUSE,
 * since that's the only reasonable way you have of learning why Intuition
 * is suddenly sending you a stream of mouse movement events.  If you don't
 * set RELVERIFY, you'll get at least one Mouse Pc ition event.
 */
#define FOLLOWMOUSE         0x0003

/* if any of the BORDER flags are set in a gadget that's included in the
 * gadget list when a window is opened, the corresponding border will
 * be adjusted to make room for the gadget
 */
#define RIGHTBORDER         0x0010
#define LEFTBORDER          0x0020
#define TOPBORDER           0x0040
#define BOTTOMBORDER        0x0080
```

```
#define TOGGLESELECT        0x0100     /* this bit for toggle-select mode */

#define STRINGCENTER        0x0200     /* should be a StringInfo flag, but it's OK*/
#define STRINGRIGHT         0x0400     /* should be a StringInfo flag, but it's OK*/

#define LONGINT             0x0800     /* this string gadget is actually LONG Int */

#define ALTKEYMAP           0x1000     /* this string has an alternate keymap */

/* — GADGET TYPES ————————————————————— */
/* These are the gadget type definitions for the variable GadgetType
 * gadget number type MUST start from one.  NO TYPES OF ZERO ALLOWED.
 * first comes the mask for gadget flags reserved for gadget typing
 */
#define GADGETTYPE     0xFC00     /* all gadget global type flags (padded) */
#define SYSGADGET      0x8000     /* 1 = SysGadget, 0 = AppliGadget */
#define SCRGADGET      0x4000     /* 1 = ScreenGadget, 0 = WindowGadget */
#define GZZGADGET      0x2000     /* 1 = gadget for GIMMEZEROZERO borders */
#define REQGADGET      0x1000     /* 1 = this is a requester gadget */
/* system gadgets */
#define SIZING         0x0010
#define WDRAGGING      0x0020
#define SDRAGGING      0x0030
#define WUPFRONT       0x0040
#define SUPFRONT       0x0050
#define WDOWNBACK      0x0060
#define SDOWNBACK      0x0070
#define CLOSE          0x0080
/* application gadgets */
#define BOOLGADGET     0x0001
#define GADGET0002     0x0002
#define PROPGADGET     0x0003
#define STRGADGET      0x0004


/* ======================================================== */
/* === Image ============================================== */
/* ======================================================== */
/* This is a brief image structure for very simple transfers of
 * image data to a RastPort
 */
struct Image
{
    SHORT   LeftEdge;            /* starting offset relative to some origin */
    SHORT   TopEdge;             /* starting offsets relative to some origin */
    SHORT   Width;               /* pixel size (though data is word-aligned) */
    SHORT   Height, Depth;       /* pixel sizes */
    USHORT  *ImageData;          /* pointer to the actual word-aligned bits */
    /* tThe PlanePick and PlaneOnOff variables work much the same way as the
     * Equivalent GELS Bob variables.  It's a space-saving
     * jnechanisra for image data.  Rather than defining the image data
     * for every plane of the RastPort, you need define data only
     * jbr the planes that are not entirely zero or one.  As you
     * Refine your imagery, you will often find that most of the planes
     * j\RE just as color selectors.  For instance, if you're designing
     * $. two-color gadget to use colors two and three, and the gadget
     * i^ill reside in a five-plane display, bit-plane zero of your
     * imagery would be all ones, bit-plane one would have data that
     * describes the imagery, and bit-planes two through four would be
     * ^11 zeroes.  Using these flags allows you to avoid wasting all
     * ihat memory in this way: first, you specify which planes you
     * ^vant your data to appear in using the PlanePick variable.  For
     * $ach bit set in the variable, the next "plane" of your image
     * <jlata is blitted to the display.  For each bit clear in this
     * tariable, the corresponding bit in PlaneOnOff is examined.
     * if that bit is clear, a "plane" of zeroes will be used.
     * jf the bit is set, ones will go out instead.  So, for our example:
     *  | GadgetPlanePick = 0x02;
     *  | GadgetPlaneOnOff = 0x01;
     * l^ote that this also allows for generic gadgets, such as the
     * System gadgets, which will work in any number of bit-planes.
     * Note also that if you want an Image that is only a filled
     * ifec tangle, you can get this by setting PlanePick to zero
     * (pick no planes of data) and set PlaneOnOff to describe the pen
     * <j;olor of the rectangle.
     */
UBYTE   PlanePick,  PlaneOnOff;

    /* if the NextImage variable is not NULL, Intuition presumes that
     * |t points to another Image structure with another Image to be
     * rendered
     */
struct  Image *NextImage;
};



/* ===== IntuiMessage ================================================== */
/* ===================================================================== */
struct IntuiMessaige
{
struct      Message ExecMessage;

    /* the Class bits correspond directly with the ID CMP Flags, except for the
     * special bit LONELYMESSAGE (defined below)
     */
ULONG   Class;
```

```
        /* the Code field is for special values like MENU number */
USHORT  Code;

        /* the Qualifier field is a copy of the current InputEvent's Qualifier */
USHORT  Qualifier;

        /* IAddress contains particular addresses for Intuition functions, like
         * the pointer to the gadget or the screen
         */
APTR    IAddress;

        /* When getting mouse movement reports, any event you get will have the
         * the mouse coordinates in these variables, the coordinates are relative
         * to the upper left corner of your window (GIMMEZEROZERO notwithstanding)
         */
SHORT   MouseX, MouseY;

        /* The time values are copies of the current system clock time.  Micros
         * are in units of microseconds, seconds are in seconds.
         */
ULONG   Seconds, Micros;

        /* The IDCMPWindow variable will always have the address of the window of
         * this IDCMP
         */
struct  Window *IDCMPWindow;

        /* system-use variable */
struct  IntuiMessage *SpecialLink;
};


/* --- IDCMP Classes ------------------------------------------------------ */
#define SIZEVERIFY          0x00000001
#define NEWSIZE             0x00000002
#define REFRESHWINDOW       0x00000004
#define MOUSEBUTTONS        0x00000008
#define MOUSEMOVE           0x00000010
#define GADGETDOWN          0x00000020
#define GADGETUP            0x00000040
#define REQSET              0x00000080
#define MENUPICK            0x00000100
#define CLOSEWINDOW         0x00000200
#define RAWKEY              0x00000400
#define REQVERIFY           0x00000800
#define REQCLEAR            0x00001000
#define MENUVERIFY          0x00002000
#define NEWPREFS            0x00004000
#define DISKINSERTED        0x00008000
#define DISKREMOVED         0x00010000
#define WBENCHMESSAGE       0x00020000
```

```
#define ACTJVEWINDOW        0x00040000
#define INACTIVEWINDOW      0x00080000
#define DELTAMOVE           0x00100000
#define VANILLAKEY          0x00200000
#define INTUITICKS          0x00400000
/* NOTEZ-B}EN:0x80000000 is reserved for internal use   */

/* The IDCMP Flags do not use this special bit, which is cleared when
 * Intuition ssnds its special message to the task, and set when Intuition
 * gets its message back from the task.  Therefore, I can check here to
 * find out fast whether or not this message is available for me to send.
 •/
#define LONELYMESSAGE       0x80000000


/* — IDCMP Codes————————————————————————*/
/* This group of codes is for the MENUVERIFY function */
#define MENtJHOT          0x0001      /* IntuiWants verification or MENUCANCEL */
#define MENUCANCEL        0x0002      /* HOT Reply of this cancels Menu operation */
#define MENfJWAITING      0x0003      /* Intuition simply wants a ReplyMsgQ ASAP */

/* This group of codes is for the WBENCHMESSAGE messages */
#define WBENCHOPEN        0x0001
#define WBENCHCLOSE       0x0002



/* ====================================================================== */
/* ======= IntuiText ==================================================== */
/* ====================================================================== */
/* IntuiText ijs a series of strings that start with a screen location
 * (always relative to the upper left comer of something) and then the
 * text of the string.  The text is null-terminated.
 */
struct IntuiText
{
   UBYTE    FrontPen, BackPen;       /* the pen numbers for the rendering */
   UBYTE    DrawMode;                /* the mode for rendering the text */
   SHORT    teftEdge;                /* relative start location for the text */
   SHORT    iFopEdge;                /* relative start location for the text */
   struct   iTextAttr *ITextFont;    /* if NULL, you accept the default */
   UBYTE    *IText;                  /* pointer to null-terminated text */
   struct   JntuiText *NextText;     /* continuation to TxWrite another text */
};
```

```
/* ======================================================================= */
/* ==== Menu ============================================================= */
/* ======================================================================= */
struct Menu
{
   struct    Menu *NextMenu;        /* same level */
   SHORT     LeftEdge, TopEdge;     /* position of the select box */
   SHORT     Width, Height;         /* dimensions of the select box */
   USHORT    Flags;                 /* see flag definitions below */
   BYTE      *MenuName;             /* text for this Menu Header */
   struct    MenuItem *FirstItem;   /* pointer to first in chain */

             /* These mysteriously-named variables are for internal use only */
   SHORT     JazzX, JazzY, BeatX, BeatY;
};

/* FLAGS SET BY BOTH THE APPLIPROG AND INTUITION */
#define MENUENABLED   0x0001        /* whether or not this menu is enabled */

/* FLAGS SET BY INTUITION */
#define MIDRAWN       0x0100        /* this menu's items are currently drawn */



/* ======================================================================= */
/* ==== MenuItem ========================================================= */
/* ======================================================================= */
struct MenuItem
{
   struct    MenuItem *NextItem;    /* pointer to next in chained list */
   SHORT     LeftEdge, TopEdge;     /* position of the select box */
   SHORT     Width, Height;         /* dimensions of the select box */
   USHORT    Flags;                 /* see the defines below */

   LONG      MutualExclude;         /* set bits mean this item excludes that */
   APTR      ItemFill;              /* points to Image, IntuiText, or NULL */

             /* When this item is pointed to by the cursor and the items highlight
              * mode HIGHIMAGE is selected, this alternate image will be displayed
              */
   APTR      SelectFill;            /* points to Image, IntuiText, or NULL */

   BYTE      Command;               /* only if appliprog sets the COMMSEQ flag */

   struct    MenuItem *SubItem;     /* if non-zero, this item has a subitem */

             /* The NextSelect field represents the menu number of next selected
              * item (when user has drag-selected several items)
              */
   USHORT    NextSelect;
};
```

```
/* FLAGS SET BY THE APPLIPROG */
#define CHECKIT         0x0001    /* whether to check this item if selected */
#define ITEMTEXT        0x0002    /* set if textual, clear if graphical item •/
#define COMMSEQ         0x0004    /* set if there's an command sequence */
#define MENUTOGGLE      0x0008    /* set to toggle the check of a menu item */
#define ITEMENABLED     0x0010    /* set if this item is enabled */

/* these are the SPECIAL HIGHLIGHT FLAG state meanings •/
#define HIGHFLAGS       0x0000    /* see definitions below for these bits */
#define HIGHIMAGE       0x0000    /* use the user's "select image" */
#define HIGHCOMP        0x0040    /* highlight by complementing the select box */
#define HIGHBOX         0x0080    /* highlight by "boxing" the selectbox */
#define HIGHNONE        0x0000    /• don't highlight */

/* FLAGS SET BY BOTH APPLIPROG AND INTUITION */
#define CHEpKED          0x0100    /* if CHECKIT, then set this when selected */

/* FLAGS SET BY INTUITION */
#define ISDRAWN         0x1000    /* this item's subs are currently drawn */
#define HIGHITEM        0x2000    /* this item is currently highlighted */
#define MENUTOGGLED     0x4000    /* this item was already toggled */



/* ============================================================= */
/* ==== NewScreen ============================================== */
/* ============================================================= */

struct NewScreen

  SHORT    LeftEdge, TopEdge, Width, Height, Depth;       /* screen dimensions */
  UBYTE    DetailPen, BlockPen;             /* for bar/border/gadget rendering */

  USHORT   ViewModes;                       /* the modes for the ViewPort (and View) */

  USHORT   Type;                            /* the screen type (see defines below) */

  struct    frextAttr *Font;                /* this screen's default text attributes */

  UBYTE     *DefaultTitIe;                  /* the default title for this screen */

  struct    Gadget ^Gadgets;                /* not used; should be NULL*/

           /* If you are opening a CUSTOMSCREEN and already have a BitMap
            * that you want used for your screen, you set the flags CUSTOMBITMAP in
            * the Types variable and you set this variable to point to your BitMap
            * structure.  The structure will be copied into your screen structure,
               /* after which you may discard your own BitMap if you want
               */
    struct     BitMap *CustomBitMap;
};


/* ====================================================================== */
/* === NewWindow ======================================================== */
/* ====================================================================== */
struct NewWindow
{
    SHORT      LeftEdge, TopEdge;         /* screen dimensions of window */
    SHORT      Width, Height;             /* screen dimensions of window */

    UBYTE      DetailPen, BlockPen;       /* for bar/border/gadget rendering */

    ULONG      IDCMPFlags;                /* user-selected IDCMP flags */

    ULONG      Flags;                     /* see Window struct for defines */

               /* You supply a linked-list of gadgets for your window.
                * This list DOES NOT include system gadgets.  You get the standard
                * system window gadgets by setting flag-bits in the variable Flags (see
                * the bit definitions under the window structure definition)
                */
    struct     Gadget *FirstGadget;

               /* The CheckMark is a pointer to the imagery that will be used when
                * rendering MenuItems of this window that want to be checkmarked
                * if this is equal to NULL, you'll get the default imagery
                */
    struct     Image *CheckMark;

    UBYTE      *Title;                    /* the title text for this window */

               /* The Screen pointer is used only if you've defined a CUSTOMSCREEN and
                * want this window to open in it.  If so, you pass the address of the
                * custom screen structure in this variable.  Otherwise, this variable
                * is ignored and doesn't have to be initialized.
                */
    struct     Screen *Screen;

               /* SUPER_BITMAP window?  If so, put the address of your BitMap structure
                * in this variable.  If not, this variable is ignored and doesn't have
                * to be initialized
                */
    struct     BitMap *BitMap;

               /* The values describe the minimum and maximum sizes of your windows.
                * These matter only if you've chosen the WINDOWSIZING gadget option,
             * which means that you want to let the user change the size of
             * this window.  You describe the minimum and maximum sizes that the
             * window can have by setting these variables.  You can initialize
             * any one of these to zero, which will mean that you want to duplicate
             * the setting for that dimension (if MinWidth == 0, MinWidth will be
             * set to the opening width of the window).
             * You can change these settings later using SetWindowLimits().
             * If you haven't asked for a SIZING gadget, you don't have to
             * initialize any of these variables.
             */
    SHORT    MinWidth, MinHeight;    /* minimums */
    SHORT    MaxWidth, MaxHeight;    /* maximums */

            /* The type variable describes the screen in which you want this window to
             * open.  The type value can either be CUSTOMSCREEN or one of the
             * system standard screen types such as WBENCHSCREEN.  See the
             * type definitions under the Screen structure
             */
    USHORT   Type;
};



/* ======================================================================= */
/* ===== Preferences ===================================================== */
/* ======================================================================= */

/* these are the definitions for the printer configurations */
#define  FILENAME_SIZE  30                 /* Filename size */

#define  POINTERSIZE (1 + 16 + 1) * 2      /* Size of Pointer data buffer */

/* These defines are for the default font size.  These actually describe the
 * height of the default fonts.  The default font type is the topaz
 * font, which is a fixed-width font that can be used in either
 * eighty-column or sixty-column mode.  The Preferences structure reflects
 * which is currently selected by the value found in the variable FontSize,
 * which may have either of the values defined below.  These values actually
 * are used to select the height of the default font.  By changing the
 * height, the resolution of the font changes as well.
 */
#define TOPAZ_EIGHTY 8
#define TOPAZ_SIXTY 9


struct Preferences
{
            /* the default font height */
    BYTE    FontHeight;                  /* height for system default font */

            /* constant describing what's hooked up to the port */
```

```
UBYTE;   PrinterPort;                  /* printer port connection */

         /* the baud rate of the port */
USHORT   BaudRate;                     /* baud rate for the serial port   */

         /* various timing rates */
struct   timeval KeyRptSpeed;          /* repeat speed for keyboard */
struct   timeval KeyRptDelay;          /* Delay before keys repeat */
struct   timeval DoubleClick;          /* Interval allowed between clicks */

         /* Intuition Pointer data */
USHORT   PointerMatrix[POINTERSIZE];        /* Definition of pointer sprite */
BYTE     XOffset;                           /* X-Offset for active 'bit' */
BYTE     YOffset;                           /* Y-Offset for active 'bit' */
USHORT   color17;                           /                                /
USHORT   color18;                           /* Colors for sprite pointer*/
USHORT   color19;                           /***********************************/
USHORT   PointerTicks;                      /* Sensitivity of the pointer*/

         /* Workbench screen colors */
USHORT   color0;                            /***********************************/
USHORT   color1;                            /*  Standard default colors */
USHORT   color2;                            /*   Used in the Workbench */
USHORT   color3;                            /***********************************/

         /* positioning data for the Intuition View */
BYTE     ViewXOffset;                        /* Offset for top lefthand corner */
BYTE     ViewYOffset;                        /* X and Y dimensions */
WORD     ViewInitX, ViewInitY;               /* View initial offset values */

BOOL     EnableCLI;                          /* CLI availability switch */

         /* printer configurations */
USHORT   PrinterType;                              /* printer type */
UBYTE    PrinterFilename[FILENAME_SIZE];           /* file for printer */

         /* print format and quality configurations */
USHORT   PrintPitch;                   /* print pitch  */
USHORT   PrintQuality;                 /* print quality  */
USHORT   PrintSpacing;                 /* number of lines per inch */
UWORD    PrintLeftMargin;              /* left margin in characters */
UWORD    PrintRightMargin;             /* right margin in characters */
USHORT   PrintImage;                   /* positive or negative */
USHORT   PrintAspect;                  /* horizontal or vertical */
USHORT   PrintShade;                   /* b&w, half-tone, or color */
WORD     PrintThreshold;               /* darkness Ctrl for b/w dumps */

         /* print paper descriptors */
USHORT   PaperSize;                    /* paper size  */
UWORD    PaperLength;                  /* paper length in number of lines */
USHORT   PaperType;                    /* continuous or single sheet */
```

```
  BYTE      padding[50];                    /* For further system expansion */
};

/• PrinterPort */
#define PARALLEL_PRINTER  0x00
#define SERIALJPRINTER       0x01

/* BaudRate */
#define BAUD_liO          0x00
#define BAUD_300          0x01
#define BAUD_1200         0x02
#define BAUD_2400         0x03
#define BAUD_4800         0x04
#define BAUD_96OO         0x05
#defineBAUD_19200         0x06
#define BAUD3GDI          0x07

/* PaperType */ i
#define FANFOIJD          0x00
#define SINGLE            0x80

/* PrintPitch */
#define PICA              0x000
#define ELITE             0x400
#define FINE              0x800

/* PrintQuality *ty
#define DRAFT !           0x000
#define LETTER            0x100

/* PrintSpacing */
#define SIX_LPI           0x000
#define EIGHT!PI          0x200

/* Print Image *)\
#define IMAGEJ^OSITIVE       0x00
#define IMAGEJNEGATIVE       0x01

/* PrintAspect */
#define ASPECTJIORIZ   0x00
#define ASPECI^VERT    0x01

/* PrintShade */
#define SHADE_BW             0x00
#define SHADE.GREYSCALE      0x01
#define SHADE_COLOR          0x02

/* PaperSize */
#deHne USJLETTER       0x00
#define USJ.EGAL       0x10
```

```
#define NJTRACTOR        0x20
#define WJTRACTOR        0x30
#define CUSTOM           0x40

/* PrinterType */
#define CUSTOM_NAME      0x00
#define ALPHA_P_101      0x01
#define BROTHER_15XL     0x02
#define CBM_MPS1000      0x03
#define DIAB_630         0x04
#define DIAB_ADVJD25     0x05
#define DIAB_C_150       0x06
#define EPSON            0x07
#define EPSON_JX_80      0x08
#define OKMATEL20        0x09
#define QU|MEJ,P_20      0x0A
/* new printer entries, 3 October 1985 */
#define HPJLASERJET      0x0B
#define HP_LASERJET_PLUS   0x0C


/* ==== PropInfo ============================================ */

/* This is the special data required by the proportional gadget
 * typically, this data will be pointed to by the gadget variable Speciallnfo
 */
struct Proplbfo
{
  USHORT Flags;    /* general-purpose flag bits (see defines below) */

             /* You initialize the pot variables before the gadget is added to
              * the system.  Then you can look here for the current settings
              * any time, even while user is playing with this gadget.  To
              * adjust these after the gadget is added to the system, use
              * ModifyProp();  the pots are the actual proportional settings,
              * where a value of zero means zero and a value of MAXPOT means
              * that the gadget is set to its maximum setting.
              */
  USHORT HorizPot;      /* 16-bit FixedPoint horizontal quantity percentage */
  USHORT VertPot;       /* 16-bit FixedPoint vertical quantity percentage */

             /* The 16-bit FixedPoint Body variables describe what percentage of
              * the entire body of stuff referred to by this gadget is actually
              * shown at one time.  This is used with the AUTOKNOB routines,
              * to adjust the size of the AUTOKNOB according to how much of
              * the data can be seen.  This is also used to decide how far
              * to advance the pots when user hits the container of the gadget.
              * For instancy, if you were controlling the display of a 5-line
              * window of text with this gadget, and there was a total of 15
             * lines that could be displayed, you would set the VertBody value to
             *     (MAXBODY / (TotalLines / DisplayLines)) = MAXBODY / 3.
             * Therefore, the AUTOKNOB would fill 1/3 of the container, and
             * if user hits the container outside of the knob, the pot would
             * advance 1/3 (plus or minus).   If there's no body to show, or
             * the total amount of displayable info is less than the display area,
             * set the body variables to the MAX.  To adjust these after the
             * gadget is added to the system, use ModifyPropQ;
             */
    USHORT  HorizBody;          /* horizontal Body */
    USHORT  VertBody;           /* vertical Body */

            /* these are the variables that Intuition sets and maintains */
    USHORT  CWidth;             /* container width (with any relativity absoluted) */
    USHORT  CHeight;            /* container height (with any relativity absoluted) */
    USHORT  HPotRes, VPotRes;   /* pot increments */
    USHORT  lieftBorder;        /* container borders */
    USHORT  topBorder;          /* container borders */
};


/* — FLAG BITS——————————————————————————— */
#define AUTOKNOB            0x0001     /* this flag says give me the auto-knob */
#define FREEHORIZ           0x0002     /* if set, the knob can move horizontally */
#define FREEVERT            0x0004     /* if set, the knob can move vertically */
#define PROI^BORDERLESS     0x0008     /* if set, no border will be rendered •/
#define KNOBHIT             0x0100     /* set when this knob is hit */

#define KNOBHMIN            6          /* minimum horizontal size of the knob */
#define KNOBVMIN            4          /* minimum vertical size of the knob */
#define MAXBODY             OxFFFF     /• maximum body value */
#defineMAXPOT               OxFFFF     /* maximum pot value */


/* ==========
/* ==== Remember ==================================================== */
/* ================================================================== */
/* This structure is used for remembering what memory has been allocated to
 * date by a given routine, so that a premature abort or systematic exit
 * can deallocate memory cleanly, easily, and completely
 */
struct Remember
{
    struct      Remember *NextRemember;
    ULONG       RememberSize;
    UBYTE       *Memory;
};
```

```
/* ====== Requester ================================================== */
/* ================================================================== */
struct Requester
{
             /* The ClipRect and BitMap are used for rendering the requester */
  struct     Requester *OlderRequest;
  SHORT      LeftEdge, TopEdge;        /* dimensions of the entire box */
  SHORT      Width, Height;            /* dimensions of the entire box */
  SHORT      RelLeft, RelTop;          /* for pointer relativity offsets */

  struct     Gadget *ReqGadget;        /* pointer to a list of gadgets */
  struct     Border *ReqBorder;        /* the box's border */
  struct     IntuiText *ReqText;       /* the box's text */
  USHORT     Flags;                    /* see definitions below */

             /* pen number for back-plane fill before draws */
  UBYTE      BackFill;
             /* Layer in place of clip rect  */
  struct     Layer *ReqLayer;

  UBYTE      ReqPad1[32];

             /* If the BitMap plane pointers are non-zero, this tells the system
              * that the image comes predrawn (if the appliprog wants to define
              * its own box, in any shape or size it wants!);  this is OK by
              * Intuition as long as there's a good correspondence between
              * the image and the specified gadgets
              */
  struct     BitMap *ImageBMap;        /* points to the BitMap of PREDRAWN imagery */
  struct     Window *RWindow;          /* added; points back to window */
  UBYTE      ReqPad2[36];
};


/* FLAGS SET BY THE APPLIPROG */
#define POINTREL              0x0001      /* if POINTREL set,TopLeft is relative to pointer*/
#define PREDRAWN              0x0002      /* if ReqBMap points to predrawn requester imagery */
/* FLAGS SET BY BOTH THE APPLIPROG AND INTUITION */

/* FLAGS SET BY INTUITION */
#define REQOFFWINDOW          0x1000      /* part of one of the gadgets was off-window */
#define REQACTIVE             0x2000      /* this requester is active */
#define SYSREQUEST            0x4000      /* this requester caused by system */
#define DEFERREFRESH          0x8000      /* this requester stops a refresh broadcast */
```

```
/* ======================================================================== */
/* === Screen ============================================================= */
/* ======================================================================== */
struct Screen
{
    struct    Screen *NextScreen;       /* linked list of screens */
    struct    Window *FirstWindow;      /* linked list of screen's windows */

    SHORT     LeftEdge, TopEdge;        /* parameters of the screen */
    SHORT     Width, Height;            /* parameters of the screen */

    SHORT     MouseY, MouseX;           /* position relative to upper left */

    USHORT    Flags;                    /* see definitions below */

    UBYTE     *Title;                   /* null-terminated title text */
    UBYTE     *DefaultTitle;            /* for windows without ScreenTitle */

              /* Bar sizes for this screen and all windows in this screen */
    BYTE      BarHeight, BarVBorder, BarHBorder, MenuVBorder, MenuHBorder;
    BYTE      WBorTop, WBorLeft, WBorRight, WBorBottom;

    struct    TextAttr *Font;            /* this screen's default font */

              /* the display data structures for this screen */
    struct    ViewPort ViewPort;         /* describing the screen's display */
    struct    RastPort RastPort;         /* describing screen rendering */
    struct    BitMap BitMap;             /* auxiliary graphexcess baggage */
    struct    Layer_Info LayerInfo;      /* each screen gets a LayerInfo */

    struct    Gadget *FirstGadget;       /* only system gadgets are supported */

    UBYTE     DetailPen, BlockPen;       /* for bar/border/gadget rendering */

              /* The following variable(s) are maintained by Intuition to support the
               * DisplayBeep() color-flashing technique.
               */
    USHORT    SaveColor0;

              /* This layer is for the screen and menu bars */
    struct    Layer *BarLayer;

    UBYTE     *ExtData;

    UBYTE     *UserData;                 /* general-purpose pointer to user data extension */
};


/* --- FLAGS SET BY INTUITION ----------------------------------------- */
/* The SCREENTYPE bits are reserved for describing various screen types
 * available under Intuition.
 */
#define SCREENTYPE          0x000F          /* all the screens types available */
/* — the definitions for the screen type ———————————————— */
#define WBENCHSCREEN        0x0001          /* Ta Da!  The Workbench */
#define CUSTOMSCREEN        0x000F          /* for that special look */
#define SHOWTITLE           0x0010          /* this gets set by a call to ShowTitle()*/
#define BEEPING             0x0020          /* set when screen is beeping */
#define CUSTOMBITMAP        0x0040          /* if you are supplying your own BitMap */



/* ======================================================================= */
/* =========== StringInfo ================================================ */
/* ======================================================================= */
/* this is the special data required by the string gadget
 * typically, this data will be pointed to by the gadget variable SpecialInfo
 */
struct StringInfo
{
            /* you initialize these variables, and then Intuition maintains them */
   UBYTE    *Buffer;              /* the buffer containing the start and final string */
   UBYTE    *UndoBuffer;          /* optional buffer for undoing current entry */
   SHORT    BufferPos;            /* character position in buffer */
   SHORT    MaxChars;             /* max number of chars in buffer (including NULL) */
   SHORT    DispPos;              /* buffer position of first displayed character */

            /* Intuition initializes and maintains these variables for you */
   SHORT    UndoPos;              /* character position in the undo buffer */
   SHORT    NumChars;             /* number of characters currently in buffer */
   SHORT    DispCount;            /* number of whole characters visible in container */
   SHORT    CLeft, CTop;          /* topleft offset of the container */
   struct   Layer *LayerPtr;      /* the RastPort containing this gadget */

            /* You can initialize this variable before the gadget is submitted to
             * Intuition, and then examine it later to discover what integer
             * the user has entered (if the user never plays with the gadget,
             * the value will be unchanged from your initial setting)
             */
   LONG     LongInt;

            /* If you want this gadget to use your own console keymapping, you
             * set the ALTKEYMAP bit in the Activation flags of the gadget, and then
             * set this variable to point to your keymap.  If you don't set the
             * ALTKEYMAP, you'll get the standard ASCII keymapping.
             */
   struct   KeyMap *AltKeyMap;
};
```

```
/* ------------------------------------------------------------------------ */
/* ======================================================================== */
struct Window
{
   struct     Window *NextWindow;    /* for the linked list in a screen */

   SHORT      LeftEdge, TopEdge;     /* screen dimensions of window */
   SHORT      Width, Height;         /* screen dimensions of window */

   SHORT      MouseY, MouseX;        /* relative to upper left of window */

   SHORT      MinWidth, MinHeight;   /* minimum sizes */
   SHORT      MaxWidth, MaxHeight;   /* maximum sizes */

   ULONG      Flags;                 /* see below for defines */

   struct     Menu *MenuStrip;       /* the strip of menu headers */

   UBYTE      *Title;                /* the title text for this window */

   struct     Requester *FirstRequest; /* all active requesters */

   struct     Requester *DMRequest;  /* double-click requester */

                                     /* count of reqs blocking window */
   SHORT      ReqCount;

   struct     Screen *WScreen;       /* this window's screen */
   struct     RastPort *RPort;       /* this window's very own RastPort */

              /* The border variables describe the window border.  If you specify
               * GIMMEZEROZERO when you open the window, then the upper-left of the
               * ClipRect for this window will be upper-left of the BitMap (with correct
               * offsets when in SuperBitMap mode; you MUST select GIMMEZEROZERO when
               * using SuperBitMap).  If you don't specify ZeroZero, then you save
               * memory (no allocation of RastPort, Layer, ClipRect and associated
               * bitmaps), but you also must offset all your writes by BorderTop,
               * BorderLeft and do your own mini-clipping to prevent writing over the
               * system gadgets
               */
   BYTE       BorderLeft, BorderTop, BorderRight, BorderBottom;
   struct     RastPort *BorderRPort;


              /* You supply a linked list of gadgets for your window.
               * This list DOES NOT include system gadgets.  You get the standard
               * window system gadgets by setting flag-bits in the variable Flags (see
               * the bit definitions below)
               */
   struct     Gadget *FirstGadget;
```

```
             /* these arc for opening/closing the windows */
struct       Window *Parent, *Descendant;

             /* Sprite data information for your own pointer;
              * set these AFTER you open the window by calling SetPointerQ
              */
USHORT       *Pointer;                  /* sprite data */
BYTE         PtrHeight;                 /* sprite height (not including sprite padding) */
BYTE         PtrWidth;                  /* sprite width (must be less than or equal to 16) */
BYTE         XOffset, YOffset;          /• sprite offsets */

             /* The IDCMP Flags and user's and Intuition's message ports •/
ULONG        IDCMPFlags;                /* user-selected flags */
struct       MsgPort *UserPort, *WindowPort;
struct       IntuiMessage *MessageKey;

UBYTE        DetailPen, BlockPen;       /* for bar/border/gadget rendering •/

             /* The CheckMark is a pointer to the imagery that will be used when
              * rendering Menultems of this window that want to be checkmarked
              * if this is equal to NULL, you'll get the default imagery
              */
struct   Image   *   CheckMark;

UBYTE        *ScreenTitIe;              /* if non-null, screen title when window is active*/

             /* These variables have the mouse coordinates relative to the
              * inner window of GIMMEZEROZERO windows.  This is compared with the
              * MouseX and MouseY variables, which contain the mouse coordinates
              * relative to the upper left corner of the window, GIMMEZEROZERO
              * notwithstanding
              */
SHORT        GZZMouseX;
SHORT        GZZMouseY;
             /* These variables contain the width and height of the inner window of
              * GIMMEZEROZERO windows.
              */
SHORT        GZZWidth;
SHORT        GZZHeight;

UBYTE        *ExtData;
BYTE         *UserData;                 /* general-purpose pointer to user data extension */

             /* This pointer keeps a duplicate of what
              * Window .RPort->Layer is ..supposed to be pointing at.
              */
struct       Jjayer *WLayer;
};
```

```
/* — FLAGS REQUESTED (NOT DIRECTLY SET THOUGH) BY THE APPLIPROG _— */
#define WINDOWSIZING        0x0001    /* include sizing system-gadget? */
#define WINDQWDRAG          0x0002    /* include dragging system-gadget? */
#define WINDOWDEPTH         0x0004    /* include depth arrangement gadget? */
#define WINDOWCLOSE         0x0008    /* include close-box system-gadget? */

#define SIZEBRIGHT          0x0010    /* size gadget uses right border */
#define SIZEBBOTTOM         0x0020    /* size gadget uses bottom border */

/* — refresh modes————————————————————— */
/* combinationg of the REFRESHBITS select the refresh type */
#define REFRESHBITS         0x00C0
#define SMARtJREFRESH       0x0000
#define SIMPLBLREFRESH      0x0040
#define SUPER^BITMAP        0x0080
#define OTHER_REFRESH       0x00C0

#define BACKDROP            0x0100    /* this is an ever-popular Backdrop window */

#define REPORTMOUSE         0x0200    /* set this to hear about every mouse move */

#define GMMEfZEROZERO       0x0400    /* make extra border stuff */

#define BORDERLESS          0x0800    /* set this to get a window sans border */

#define ACTIVATE            0x1000    /* when window opens, it's the active one */

/* FLAGS SET BY INTUITION */
#deiine WEMDOWACTIVE        0x2000    /* this window is the active one */
#define INREQUEST           0x4000    /* this window is in request mode */
#define MENU$TATE           0x8000    /* this window is active with its menus on */

/* — Other uset flags ————————————————————— */
#define RMBTRAP             0x00010000    /* Catch RMB events for your own */
#define NOCA^EREFRESH       0x00020000    /* not to be bothered with REFRESH */

/* — Other Intuition Flags————————————————— */
#define WINDOWREFRESH       0x01000000    /* window is currently refreshing */
#deHne WBENCHWINDOW         0x02000000    /* WorkBench tool ONLY window */
#define WINDOWTICKED        0x04000000    /* only one timer tick at a time */

#define SUPERjJJNUSED       OxFCFCOOOO    /* bits of Flag unused yet */

/* — see struct IntuiMessage for the IDCMP Flag definitions—————*/
/* ================================================================== */
/* =====  Miscellaneous  =========================================== */
/* ================================================================== */

/* =====  MACROS  ================================================== */
#define MENUNUM(n)      (n & OxlF)
#define ITEMNUM(n)      ((n >> 5)&OxOO3F)
#define SUBNUM(n)       ((n >> 11) & OxOOlF)

#define SHIFTMENU(n)    (n & OxlF)
#define SHIFTITEM(n)    ((n&0x3F) << 5)
#define SHIFTSUB(n)     ((n&OxlF) << 11)

/• =====  MENU STUFF  =============================================== */
#define NOMENU     OxOOlF
#define NOITEM     0x003F
#define NOSUB      OxOOlF
#define MENUNULL   OxFFFF


/• =====  ==RJ='s peculiarities  ==================================== */
#define FOREVER    for(;;)
#define SIGN(x)( ((x) > 0) - ((x) < 0))
#define NOT!   !

/* These defines are for the COMMSEQ and CHECKIT menu stufif.  If CHECKIT,
 * I'll use a generic width (for all resolutions) for the CheckMark.
 * If COMM9EQ, likewise I'll use this generic stuff
 */
#define CHECKWIDTH          19
#define COMMWIDTH           27
#define LOWCHECKWIDTH       13
#define LOWCOMMWIDTH        16


/* These are ihe AlertNumber defines, if you are calling Display Alert()
 * the AlertNumber you supply must have the ALERT__TYPE bits set to one
 * of these patterns
 */
#define ALERTJTYPE          0x80000000
#define RECOVERY_ALERT      0x00000000    /* the system can recover from this */
#define DEADEND_ALERT       0x80000000    /* no recovery possible, this is it */


/* When you're defining IntuiText for the positive and negative gadgets
 * created by a call to AutoRequestQ, these defines will get you
 * reasonable-looking text.  The only field without a define is the IText
 * field; you decide what text goes with the gadget
 */
#define AUTOFRONTPEN -»     0
#define AUTOBACKPEN         1
```

```
#define AUTODRAWMODE     JAM2
#define AUTOLEFTEDGE     6
#define AUTOTOPEDGE      3
#define AUTOITEXTFONT    NULL
#define AUTONEXTTEXT     NULL


/* — RAWMOUSE Codes and Qualifiers (Console OR IDCMP)————————— */
#define SELECTUP        (IECODEJLBUTTON | IECODE.UP^PREFIX)
#define SELECTDOWN      (lECODE.LBUTTON)
^define MENUUP          (IECODE.RBUTTON | IECODE_UP_PREFDC)
#define MENUDOWN        (IECODE.RBUTTON)
#define ALTLEFT         (IEQUALIFIER_LALT)
#define ALTRIGHT        (IEQUALIFIER_RALT)
#define AMIGALEFT       (IEQUALIFIER.LCOMMAND)
#define AMIGARIGHT      (IEQUALIFIER_RCOMMAND)
#define AMGAKEYS        (AMIGALEFT | AMIGARIGHT)

#define CURSORUP        0x4C
#define CURSORLEFT      0x4F
#define CURSORRIGHT     0x4E
#define CURSORDOWN      0x4D
#define KEYCODE.Q       0x10
#define KEYCODE.X       0x32
#define KEYCODE_N       0x36
#define KEYOODE.M       0x37


#endif
```

> $24.95 FPT USA

Ami[illegible] Reference Series

Amiga [illegible]I Reference Manual

The Amiga Computer is an exciting new high-performance microcomputer with superb graphics, sound, and multitasking capabilities. Its technologically advanced hardware, designed around the Motorola 68000 microprocessor, includes three sophisticated custom chips that control graphics, audio, and peripherals. The Amiga's unique system software is contained in 192K of read-only memory (ROM), providing programmers with unparalleled power, flexibility, and convenience in designing and creating programs.

The AMIGA INTUITION REFERENCE MANUAL, written by the technical staff at Commodore-Amiga, Inc., is a complete description of Intuition, the Amiga user interface. It includes:

- a general introduction to Intuition's features
- a complete listing of the components of Intuition, including specifications for the data structure of each component and a brief summary of the function calls that affect that component
- important programming style guidelines
- a glossary of key terms

For the serious programmer working in assembly language, C, or Pascal who wants to create application programs that take advantage of the Amiga's impressive capabilities and are consistent and easy to use, the AMIGA INTUITION REFERENCE MANUAL is an essential reference.

Written by the technical staff at Commodore-Amiga, Inc., who designed the Amiga, the AMIGA INTUITION REFERENCE MANUAL is the definitive source of information on the user interface built into this revolutionary microcomputer.

Cover design by Marshall Henrichs
Cover photograph by Jack Haeger

Addison-Wesley Publishing Company, Inc. ISBN 0-201-11076-3