

Libraries_Manual

COLLABORATORS

	<i>TITLE :</i> Libraries_Manual		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 18, 2024	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Libraries_Manual	1
1.1	Amiga® RKM Libraries: 10 Intuition Mouse and Keyboard	1
1.2	10 Intuition Mouse and Keyboard / The Mouse	1
1.3	10 / The Mouse / Intuition's Use of Mouse Events	2
1.4	10 // Intuition's Use of Mouse Events / Select Button	3
1.5	10 // Intuition's Use of Mouse Events / Menu Button	4
1.6	10 / The Mouse / Mouse Messages	4
1.7	10 / The Mouse / Mouse Usage Example	6
1.8	10 Intuition Mouse and Keyboard / The Pointer	6
1.9	10 / The Pointer / Pointer Position	6
1.10	10 / The Pointer / Custom Pointer	7
1.11	10 // Custom Pointer / The Sprite Data Structure	8
1.12	10 Intuition Mouse and Keyboard / The Keyboard	9
1.13	10 / The Keyboard / Keyboard Control of the Pointer	10
1.14	10 / The Keyboard / Intuition Keyboard Shortcuts	11
1.15	10 / The Keyboard / Menu Shortcuts	12
1.16	10 / The Keyboard / Amiga Qualifiers	12
1.17	10 Intuition Mouse and Keyboard / Function Reference	13

Chapter 1

Libraries_Manual

1.1 Amiga® RKM Libraries: 10 Intuition Mouse and Keyboard

In the Intuition system, the mouse is the normal method of making selections and the keyboard is used for entering character data. This section describes how users employ the mouse to interact with the system and how to arrange for a program to use the mouse. It also describes the use of the keyboard, both as a character input device and as an alternate method of controlling the mouse pointer.

The Mouse The Pointer The Keyboard Function Reference

1.2 10 Intuition Mouse and Keyboard / The Mouse

The Amiga mouse is a small, hand-held input device connected to the Amiga by a flexible cable. The user can input horizontal and vertical coordinates with the mouse by sliding it around on a smooth surface. This movement causes the repositioning of a pointer on the display; whenever the mouse is moved the pointer moves, and in the same direction.

The mouse also provides two or three input keys, called mouse buttons, that allow the user to input information to the computer. The basic activities the user can perform with the mouse are shown below.

Action -----	Explanation -----
Moving the Mouse	Sliding the body of the mouse over a surface, such as a desk top.
Pressing a button	Pushing down a mouse button (which is released at some later time).
Clicking a button	Quickly pressing and releasing a mouse button.
Double clicking a button	Clicking a button twice in a short period of time.

Dragging Pressing a button and moving the mouse while the button is held down. The drag operation is completed by releasing the button.

Table 10-1: Mouse Activities

The action associated with mouse button presses can occur when the button is first pressed, or while the button is held down, or when the button is released. As an example of this, consider the drag gadget of a window. When the select button of the mouse is first pressed an outline representing the window frame is drawn. While the button is held down the outline remains, and it moves with the pointer as the mouse is moved. When the button is released, the outline is erased and the window takes its new position.

Intuition's Use of Mouse Events	Mouse Usage Example
Mouse Messages	

1.3 10 / The Mouse / Intuition's Use of Mouse Events

When the mouse is moved or its buttons are pressed, the system generates input events that represent the actions. The input events are taken from the input chain by Intuition when the active window requires the events. Note that only input for a specific window will be affected by changes in that window's IDCMP flags.

Most events generated by the user with the mouse are used by Intuition.

As the user moves the mouse, Intuition changes the position of its pointer. The Intuition pointer moves around the entire video display, mimicking the user's movement of the mouse. The user points at an object by positioning the hot spot of the pointer over the object. The hot spot is the active part of the pointer image; the hot spot for Intuition's default pointer is the pixel at the tip of the arrow.

After pointing to an object, the user can perform some action on that object by selecting it with one of the mouse buttons. These can include any of the actions specified above, such as dragging or double clicking.

The left mouse button is generally used for selection, while the right mouse button is most often used for information transfer. The terms selection and information are intentionally left open to some interpretation, as it is impossible to imagine all the possible uses for the mouse buttons.

The selection/information paradigm can be crafted to cover most interaction between the user and an application. When using the mouse, the application should emphasize this model. It will help the user to understand and remember the mouse control of the application.

Applications that handle mouse button events directly, bypassing the menu

and gadget systems, should use the same selection/information model used by Intuition.

Select Button Menu Button

1.4 10 // Intuition's Use of Mouse Events / Select Button

When the user presses the left, or select button, Intuition examines the state of the system and the position of the pointer. This information is used to decide whether or not the user is trying to select some object, operation, or option. For example, the user positions the pointer over a gadget and then presses the left button to select that gadget. Alternatively, the user can position the pointer over a window and press the select button to activate the window. The pointer is said to be over an object when the pointer's hot spot is positioned within the selection region of the object.

A number of other common techniques involving the select button are available. They include:

Drag Select

Multiple objects or an extended area may be selected by dragging the mouse over a range with the select button held down. For instance, in Release 2, multiple icons can be selected in a Workbench window by pressing the select button while the pointer is over the background of the window (not an icon or a system gadget) and then moving the mouse with the select button held down. A selection rectangle will be displayed and all icons within the rectangle will be selected. Similarly, the user may highlight blocks of text in a console window by pressing the select button over the first desired character and dragging the mouse to the last desired character while holding the button down.

Multi-Select or Shift Select

Another way to select multiple objects or an extended area is through the shift select technique. First, select the first member of the group of objects in the normal way. Additional objects can be added to the group by holding down the Shift key while the select button is pressed. This technique works with Workbench icons, where icons may be added one-at-a-time to the list of selected icons; and with text in a console window, where the selected text is extended to include the new position. Note that text need not operate this way, and the application may allow multiple discrete blocks to be selected at any given time.

Cancel Drag Operation

Both drag select and the dragging of individual objects may often be canceled by pressing the right mouse button before completing the drag operation (before releasing the select button). Examples of this include window dragging and sizing, and positioning of Workbench icons.

1.5 10 // Intuition's Use of Mouse Events / Menu Button

The right mouse button is used to initiate and control information gathering processes. Intuition uses this button most often for menu operations.

For most active windows, pressing the menu button will display the window's menu bar at the top of the screen. Dragging the mouse with the menu button depressed allows the user to browse through the available menus. Releasing the right mouse button over a menu item will select that item, if it is a valid choice. Additionally, the user can select multiple items by repeatedly pressing the select button while the menu button is held down.

Drag selection is also available in menu operations. When the menu system is activated, and the user has the menu button pressed, the select button may be pressed and the mouse dragged over all items to be selected. This only works if the select button is pressed after the menu button, and all items that the pointer travels over will be selected.

Double clicking the right mouse button can bring up a special requester for extended exchange of information. This requester is called the double-menu requester, because a double click of the menu button is required to reveal it, and because this requester acts like a super menu through which a complex exchange of information can take place. Because the requester is used for the transfer of information, it is appropriate that this mechanism is called up by using the right button.

The programmer should consult the Amiga User Interface Style Guide for more information on the standard uses of the mouse and its buttons.

Button activation and mouse movements can be combined to create compound instructions. For example, Intuition combines multiple mouse events when displaying the menu system. While the right button is pressed to reveal the menu items of the active window, the user can move the mouse to position the pointer and display different menu items and sub-items. Additionally, multiple presses of the left button can be used to select more than one option from the menus.

Dragging can have different effects, depending on the object being dragged. Dragging a window by the drag gadget will change the position of the window. Dragging a window by the sizing gadget will change the size of the window. Dragging a range in a Workbench window will select all of the icons in the rectangular range.

1.6 10 / The Mouse / Mouse Messages

Mouse events are broadcast to the application via the IDCMP or the console device. See the "Intuition Input and Output Methods" chapter in this book for information on the IDCMP. See the "Console Device" chapter in the Amiga ROM Kernel Reference Manual: Devices for more about the console device.

Simple mouse button activity not associated with any Intuition function

will be reported to the window as an IntuiMessage with a Class of IDCMP_MOUSEBUTTONS. The IntuiMessage Code field will be set to SELECTDOWN, SELECTUP, MIDDLEDOWN, MIDDLEUP, MENUDOWN or MENUUP to specify changes in the state of the left, middle and right buttons, respectively.

Direct select button events will not be received by the program if the select button is pressed while the pointer is positioned over a gadget or other object which uses the button event. For example, select button activity over a gadget is reported with a Class of IDCMP_GADGETDOWN or IDCMP_GADGETUP. The gadget is said to have consumed the mouse events and produced gadget events.

If the menu system is enabled, menu selections appear with a Class of IDCMP_MENUPICK. To directly receive menu button events, the application must set the flag WFLG_RMBTRAP for the window either when the window is opened or by changing the flag in a single, atomic operation. See the chapter "Intuition Windows" for more information on the flag WFLG_RMBTRAP.

The program receives mouse position changes in the event Class IDCMP_MOUSEMOVE. The MouseX and MouseY position coordinates describe the position of the mouse relative to the upper left corner of the reference window. These coordinates are always in the resolution of the screen being used, and may represent any pixel position on the screen, even though the hardware sprites can be positioned only on the even numbered pixels of a high resolution screen and on the even numbered rows of an interlaced screen. Enabling IDCMP_MOUSEMOVE messages is discussed below in the section on "The Pointer".

To get mouse movement reported as deltas (amount of change from the last position) instead of as absolute positions, set the IDCMP flag IDCMP_DELTAMOVE. When IDCMP_DELTAMOVE is set, the IDCMP_MOUSEMOVE messages received by the program will have delta values rather than absolute values. Note that IDCMP_DELTAMOVE is simply a flag used to modify the behavior of IDCMP_MOUSEMOVE, and that no messages of class IDCMP_DELTAMOVE are ever sent.

Each window has a queue limit for the number of IDCMP_MOUSEMOVE messages waiting on its IDCMP at any given time. If the number of mouse move messages waiting at the IDCMP is equal to the queue limit, then Intuition will discard additional IDCMP_MOUSEMOVE messages until the application replies to one of the queued mouse move messages. The default queue limit for mouse move messages is five.

Be aware that this may cause some data loss, especially when the application is using IDCMP_DELTAMOVE, as the information contained in the discarded messages is not repeated. When using IDCMP_DELTAMOVE, this could cause the application to lose track of the actual pointer position. The application may wish to change the default mouse queue size if it is unable to reply to messages queued at the IDCMP for an extended period. The mouse queue can be set when the window is opened by using the WA_MouseQueue tag, and may later be modified using the SetMouseQueue() call. Note that the actual mouse position is always available to the application through the Window structure MouseX and MouseY.

1.7 10 / The Mouse / Mouse Usage Example

The example program below shows the use of IDCMP_MOUSEBUTTONS, IDCMP_MOUSEMOVE and DoubleClick(). DoubleClick() is used to test the interval between two times and determine if the interval is within the user specified time for double clicking as set in the Preferences Input editor.

```
BOOL DoubleClick( unsigned long sSeconds, unsigned long sMicros,
                  unsigned long cSeconds, unsigned long cMicros );
```

The sSeconds and sMicros arguments specify a timestamp value describing the start of the double click time interval to be tested. The cSeconds and cMicros arguments specify a timestamp value describing the end of the double click time interval to be tested.

DoubleClick() returns TRUE if the time interval was short enough to qualify as a double-click. A FALSE return indicates that the time interval between presses took too long. The button presses should be treated as separate events in that case.

mousestest.c

1.8 10 Intuition Mouse and Keyboard / The Pointer

The system provides a pointer to allow the user to make selections from menus, choose gadgets, and so on. The user may control the pointer with a mouse, the keyboard cursor keys or some other type of controller. The specific type of controller is not important, as long as the proper types of input events can be generated.

The pointer is associated with the active window and the input focus. The active window controls the pointer imagery and receives the input stream from the mouse. The pointer and mouse may be used to change the input focus by selecting another window.

Pointer Position Custom Pointer Pointer Example

1.9 10 / The Pointer / Pointer Position

There are two ways to determine the position of the pointer: by direct examination of variables in the window structure at any time, and by examining messages sent by Intuition which inform the application of pointer movement. The pointer coordinates are relative to the upper left corner of the window and are reported in the resolution of the screen, even though the pointer's visible resolution is always in low-resolution pixels (note that the pointer is actually a sprite).

The MouseX and MouseY fields of the Window structure always contain the current pointer x and y coordinates, whether or not the window is the active one. If the window is a GimmeZeroZero window, the variables GZZMouseX and GZZMouseY in the Window structure contain the position of

the mouse relative to the upper left corner of the inner window.

If the window is receiving mouse move messages, it will get a set of x,y coordinates each time the pointer moves. To receive messages about pointer movements, the `WFLG_REPORTMOUSE` flag must be set in the Window structure. This flag can be set when the window is opened. The flag can also be modified after the window is open by calling `ReportMouse()`, however C programmers should avoid this function. `ReportMouse()` has problems due to historic confusion about the ordering of its C language arguments. Do not use `ReportMouse()` unless you are programming in assembler. C programmers should set the flag directly in the Window structure using an atomic operation.

Most compilers generate atomic code for operations such as `mywindow->flags |= WFLG_REPORTMOUSE` or `mywindow->flags &= ~WFLG_REPORTMOUSE`. If you are unsure of getting an atomic operation from your compiler, you may wish to do this operation in assembler, or bracket the code with a `Forbid()/Permit()` pair.

After the `WFLG_REPORTMOUSE` flag is set, whenever the window is active it will be sent an `IDCMP_MOUSEMOVE` messages each time the pointer position changes. The window must have the IDCMP flag `IDCMP_MOUSEMOVE` set to receive these messages.

Mouse movements can cause a very large number of messages to be sent to the IDCMP, the application should be prepared to handle them efficiently.

Messages about pointer movements may also be activated by setting the flag `GACT_FOLLOWMOUSE` in an application Gadget structure. When this flag is set in a gadget, changes in the pointer position are reported as long as the gadget is selected by the user. These messages are also sent as `IDCMP_MOUSEMOVE` messages.

1.10 10 / The Pointer / Custom Pointer

An application can set a custom pointer for a window to replace the default pointer. This custom pointer will be displayed whenever the window is the active one.

To place a custom pointer in a window, call `SetPointer()`.

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

Set the window argument to the address of the window that is to receive this custom pointer definition. The pointer argument is the address of the data that defines the custom pointer image. The format of this data is discussed in the next section, "The Sprite Data Structure".

The height and width specify the dimensions of the pointer sprite. There is no height restriction but the width of the sprite must be less than or equal to 16.

The `xOffset` and `yOffset` 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 this is the offset of the default Intuition pointer hot spot from the top left corner of the sprite.

For instance, by specifying offsets of (0,0), the top left corner of the sprite image will be placed at the pointer position. On the other hand, specifying an xOffset of -7 (remember, sprites are 16 pixels wide) will center the sprite over the pointer position. Specifying an xOffset of -15 will place the right edge of the sprite will be over the pointer position.

Specifying the Hot Spot.

For compatibility, the application must specify that the "hot spot" of the pointer is one pixel to the left of the desired position. Changes to the pointer done by a program must compensate for this. The Preferences Pointer editor correctly handles this situation.

To remove the custom pointer from the window, call `ClearPointer()`.

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

Set the window argument to the address of the window that is to have its custom pointer definition cleared. The pointer will be restored to the default Intuition pointer imagery

`SetPointer()` and `ClearPointer()` take effect immediately if the window is active, otherwise, the change will only be displayed when the window is made active.

The Sprite Data Structure

1.11 10 // Custom Pointer / The Sprite Data Structure

To define the pointer, set up a sprite data structure (sprites are one of the general purpose Amiga graphics structures). The sprite image data must be located in Chip memory, which is memory that can be accessed by the special Amiga hardware chips. Expansion, or Fast memory cannot be addressed by the custom chips. Ensure that data is in Chip memory by using the `AllocMem()` function with the `MEMF_CHIP` flag, and copying the data to the allocated space. Alternately, use the tools or flags provided by each compiler for this purpose. See the "Exec Memory Allocation" chapter for more information.

A sprite data structure is made up of words of data. In a pointer sprite, the first two words and the last two words are reserved for the system and should be set to zero. All other words contain the sprite image data.

The pointer in the example, a standard busy pointer, is sixteen lines high and sixteen pixels wide. Currently, all sprites are two bit planes deep, with one word of data for each line of each plane. The example sprite image consists of 36 words (2 planes x 18 lines = 36 words). Add to this the four reserved words of control information for a total of 40 words of data. See the example below for the complete data definition.

The sprite data words are combined to determine which color will appear at each pixel position of each row of the sprite. The first two words of image data, 0x0400 and 0x07C0, represent the top line of the sprite. The numbers must be viewed as binary numbers and combined in a bit-wise fashion. The highest bit from each word are combined to form a two bit number representing the color register for the leftmost pixel. The next two bits represent the next pixel in the row, and so on, until the low order bits from each word represent the rightmost pixel in the row.

For example:

```

Hex      Binary
----     -
Second word 0x07C0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0
First word  0x0400 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0

          \_____/ \___/ \_____/ \_____ /
            |       |       |           |
            |       |       |           |-- 00 = color 0
            |       |       |           |
            |       |       |       --- 10 = color 2
            |       |       |       |
            |       |       |       --- 11 = color 3
            |       |       |       |
            |       |       |       --- 00 = color 0

```

Pointer Color Ordering.

The first word in a line gives the least significant bit of the color register and the second word gives the most significant bit.

Sprites get their color information from the color registers much like screens do. See the Amiga Hardware Reference Manual for more information on the assignment of color registers to sprites. Note that the color number given above is added to a base number to determine the actual hardware color register.

The colors of the Intuition pointer may be changed. The Intuition pointer is always sprite 0. To change the colors of sprite 0, call the graphics library routine `SetRGB4()`.

1.12 10 Intuition Mouse and Keyboard / The Keyboard

A program can receive keyboard data through an IDCMP port by setting the IDCMP_RAWKEY flag, the IDCMP_VANILLAKEY flag or both. IDCMP_VANILLAKEY events provide for simple ASCII text and standard control keys like space, return and backspace. IDCMP_RAWKEY events provide a more complex input stream, which the program must process to generate ASCII data. IDCMP_RAWKEY returns all keycodes, both key-up and key-down, including function keys.

Keystrokes Are Not Always Paired.

Keystrokes do not always come in key-down/key-up pairs. For example, repeating keys appear as a sequence of key-down messages.

IDCMP_RAWKEY and IDCMP_VANILLAKEY may be set together. When both flags are set in the IDCMP, IDCMP_VANILLAKEY messages will be sent for keystrokes that directly map to a single ASCII value. IDCMP_RAWKEY messages will be sent for key sequences that do not map to simple values, i.e. if a key sequence does not map to an IDCMP_VANILLAKEY message, it will be sent as an IDCMP_RAWKEY message. This allows easy access to mapped characters through IDCMP_VANILLAKEY with control characters returned as IDCMP_RAWKEY. Note that the IDCMP_RAWKEY events will only return the key down events when used with IDCMP_VANILLAKEY.

When Intuition responds to an input event or sequence of events, the application will not receive those events. This happens for system shortcuts (left Amiga + key) if the system shortcut is defined, and for menu shortcuts (right Amiga + key) if the menu shortcut is defined for the active window. If the shortcut is not defined, then the appropriate key event will be sent with the proper Amiga qualifier set.

Key repeat characters have a queue limit which may be set for each window, much like the mouse queue described above. The key repeat queue limit may only be set when the window is opened using the WA_RptQueue tag, there is no function call for modifying the value after the window is open. The default queue limit for key repeat characters is three. This limit causes any IDCMP_RAWKEY, IDCMP_VANILLAKEY or IDCMP_UPDATE message with the IEQUALIFIER_REPEAT bit set to be discarded if the queue is full (IDCMP_UPDATE is discussed in the "BOOPSI" chapter). The queue is said to be full when the number of waiting repeat key messages is equal to the queue limit. Note that the limit is not per character, it is on the total number of key messages with the repeat bit set. Once the limit is reached, no other repeat characters will be posted to the IDCMP until the application replies to one of the outstanding repeat key messages. The repeat queue limit is not as dangerous as the mouse queue limit as only duplicate keystroke information is discarded, where the mouse queue limit discards information that cannot be easily reproduced.

Rawkey Keymapping Example	Menu Shortcuts
Keyboard Control of the Pointer	Amiga Qualifiers
Intuition Keyboard Shortcuts	

1.13 10 / The Keyboard / Keyboard Control of the Pointer

All Intuition mouse activities can be emulated using the keyboard, by combining the Amiga command keys with other keystrokes.

The pointer can be moved by holding down either Amiga key along with one of the four cursor keys. The mouse pointer accelerates the longer these keys are held down. Additionally, holding down either Shift key will make the pointer jump in larger increments. The pointer position may also be adjusted in very fine increments through this technique. By holding down either Amiga key and briefly pressing one of the cursor keys, the pointer may be moved one pixel in any direction.

Press the left Alt key and either one of the Amiga keys simultaneously emulates the left button of the mouse. Similarly, pressing the right Alt key and either one of the Amiga keys simultaneously emulates the right button of the mouse. These key combinations permit users to make gadget selections and perform menu operations using the keyboard alone.

1.14 10 / The Keyboard / Intuition Keyboard Shortcuts

If Intuition sees a command key sequence that means nothing to it, the key sequence is sent to the active window as usual. See the chapter on "Intuition Input and Output Methods" for how this works. This section and the next section describe what Intuition does when it recognizes certain special command key sequences.

It is recommended that programs abide by certain command key standards to provide a consistent interface for Amiga users. The Amiga User Interface Style Guide contains a complete list of the recommended standards.

There are a number of special keyboard shortcuts supported by Intuition. These involve holding down the left Amiga key and simultaneously pressing a another key. These functions allow the user to do such things as move the Workbench screen to the front using the keyboard.

Table 10-2: Intuition Keyboard Shortcuts

Keyboard Shortcut	Function Performed
left Amiga M	Move frontmost screen to back.
left Amiga N	Move Workbench screen to front.
left Amiga B	System requester cancel, or select the rightmost button in the system requester.
left Amiga V	System requester OK, or select the leftmost button in the system requester.
left Amiga + mouse select button	Screen drag from any point. By holding down the left Amiga key, the user may drag the screen with the mouse from any part of the screen or window on the screen.

About System Keyboard Shortcuts

Many of these keyboard commands may be remapped through the IControl Preferences editor. Do not rely on the values reported here.

Intuition consumes these command key sequences for its own use. That is, it always detects these events and removes them from the input stream. The application will not see the events.

1.15 10 / The Keyboard / Menu Shortcuts

Menu items and sub-items may be paired with command key sequences to associate certain characters with specific menu item selections. This gives the user a shortcut method to select frequently used menu operations, such as Undo, Cut, and Paste. Whenever the user presses the right Amiga key with an alphanumeric key, the menu strip of the active window is scanned to see if there are any command key sequences in the list that match the sequence entered by the user. If there is a match, Intuition translates the key combination into the appropriate menu item number and transmits the menu number to the application program.

Menu Shortcuts Look Like the Real Thing.

To the application it looks as if the user had selected a given menu item with the mouse. The program will receive a menu event, not a key event. For more information on menu item selection, see the "Intuition Menus" chapter.

1.16 10 / The Keyboard / Amiga Qualifiers

The Amiga keyboard has several special qualifiers which are listed in the next table. Most of these qualifiers are associated with special keys on the keyboard such as the Shift or Ctrl key. These keys are used to modify the meaning of other keys. Other qualifiers are associated with mouse button status. For a complete list of all the qualifiers, see the include file <devices/inpotevent.h>.

The Qualifier field of each IntuiMessage contains the status of all the qualifiers. An individual application should never attempt to track the state of any of the qualifier keys or mouse buttons even though key-down and key-up information may be available. Instead use the information available in the Qualifier field of the IntuiMessage structure.

Table 10-3: Keyboard Qualifiers

Qualifier Type	Key Label	Explanation
-----	-----	-----
Control	Ctrl	The IEQUALIFIER_CONTROL bit indicates that the Control key is depressed.
Amiga	Fancy A	There are two Amiga keys, one on each side of the space bar. The left Amiga key is recognized by the Qualifier bit IEQUALIFIER_LCOMMAND, and the right Amiga key by IEQUALIFIER_RCOMMAND.
Alternate	Alt	There are two separate Alt keys, one on each side of the space bar, next to the Amiga keys. These can be treated separately, if desired. The left Alt key sets the IEQUALIFIER_LALT bit and

		the right Alt key sets the IEQUALIFIER_RALT bit.
Shift	Up Arrow	There are two separate Shift keys, one above each Alt key. These can be treated distinctly, if desired. The left Shift key sets the IEQUALIFIER_LSHIFT bit and the right Shift key sets the IEQUALIFIER_RSHIFT bit.
Caps Lock	Caps Lock	The IEQUALIFIER_CAPSLOCK bit is set as long as the Caps Lock light is illuminated.
Numeric Pad		The IEQUALIFIER_NUMERICPAD bit is set for keys on the numeric keypad.
Repeat		Repeat key events are sent with the IEQUALIFIER_REPEAT bit set.
Mouse Buttons		If mouse buttons are down when the event occurs, one or more of the three bits IEQUALIFIER_LEFTBUTTON, IEQUALIFIER_MIDBUTTON or IEQUALIFIER_RBUTTON will be set.

1.17 10 Intuition Mouse and Keyboard / Function Reference

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

Table 10-4: Functions for Intuition Mouse and Keyboard

Function	Description
DoubleClick()	Test two time values for double click status.
SetPointer()	Change the Intuition pointer imagery for an open window.
ClearPointer()	Restore the default Intuition pointer imagery.
SetMouseQueue()	Change the mouse queue for an open window.
ReportMouse()	A function C programmers should not use.