Libraries

COLLABORATORS			
	TITLE : Libraries		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 23, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Libraries

1.1 Amiga® RKM Libraries: 9 Intuition Input and Output Methods

This chapter discusses the input and output (I/O) techniques used with Intuition. I/O facilities are also available through Exec's device subsystems, such as the console, serial and parallel devices. (For more information on these see the Amiga ROM Kernel Reference Manual: Devices.)

For graphical output to the Amiga's display, programs can use Intuition's drawing features or handle rendering directly through calls to the graphics library. See the three graphics library chapters in this manual for more information on display rendering. For more about Intuition's drawing features see the "Intuition Images, Line Drawing and Text" chapter.

1.2 9 Intuition Input and Output Methods / Overview of System I/O

This section provides a very simplified model of how Amiga I/O and application programs interact. The main elements of the Amiga's I/O system are shown in the diagram below. Input events begin when mouse movement is detected by the gameport device or key presses are received by the keyboard device. These and other input events are merged into a single stream by the input device, which then submits the stream to Intuition for further processing.

Figure 9-1: Amiga Input Block Diagram

The application program can receive its input from Intuition or the Console device. The application may choose to listen to neither, one or both of these input sources.

Figure 9-2: Amiga Output Block Diagram

An application's display output can go through the high level interfaces

of the console device or through the Intuition library. Additionally, display output may be sent directly to the graphics library. Notice that both the Console and Intuition call the graphics library to render to the display.

1.3 9 Intuition Input and Output Methods / Intuition Input

The Amiga has an input device to monitor all input activity. The input activity nominally includes keyboard and mouse events, but which can be extended to include other types of input signals. When the user moves the mouse, presses a mouse button or types on the keyboard, the input device detects the activity from the specific device, and constructs an InputEvent.

An InputEvent is a message describing a single event, such as the transition of a key on the keyboard from up to down.

The input device then passes the input events down a prioritized chain of input handlers, which are routines in memory that process the input events. The sequence of input events passing through this chain of input handlers is known as the input stream. Any handler linked into this chain can monitor and modify the event stream.

Each input handler may block (consume) events, allow events to pass through to the next handler in the chain or add new events to the sequence.

Other devices and programs can add input events to the input stream by sending messages to the input device. For instance, AmigaDOS is able to generate an input event whenever a disk is inserted or removed.

See the "Input Device" chapter of the Amiga ROM Kernel Reference Manual: Devices for more information on the Input device.

1.4 9 / Intuition Input / Intuition as an Input Handler

Intuition is an input handler linked into the input stream, and it monitors and modifies events that it receives. The input arrives at Intuition as a single stream of events. These events are filtered, altered, and enhanced by Intuition, then dispatched to windows as appropriate, or passed down to input handlers lower in the chain. If the active window has a console attached to it, then it can receive the input events that are still left in the stream, which can include some events that Intuition played a role in forming.

Many kinds of input event undergo little conversion by Intuition. For instance, raw keyboard events are not modified by Intuition (with the exception of a few keystrokes that have special meaning). Other events may produce differing results based on Intuition’s view of the system. For

example, when the mouse select button is pressed, the event may become a gadget down-press event, a window activation event, or it may remain a simple button press, depending on the mouse position and the arrangement of windows and screens. Still other events are consumed by Intuition, and the application is not directly notified. An example would be when the select button is pressed over a system gadget.

Intuition is also the originator of certain kinds of events. For example, a window-refreshing event is generated when Intuition discovers that part of a window is in need of redrawing. This might have resulted indirectly from some other input (for example, the user might have dragged a window), but not necessarily (the refresh might have been necessitated by a program bringing a window to the front).

1.5 9 / Intuition Input / Receiving Input Events from Intuition

There are two channels through which a window can receive events destined for it. The usual way is for the application to ask Intuition to send it messages which are based on the input event that Intuition has processed. These messages, called IntuiMessages, are standard Amiga Exec messages, and are sent to a port called an Intuition Direct Communications Message Port, or IDCMP. Every window may have an IDCMP associated with it (pointed to by Window.UserPort).

There are many classes of IntuiMessages, and the application can control which classes of events are routed to its window's port by setting the appropriate IDCMP flags. When Intuition has an event to send, but the window does not have the corresponding IDCMP flag set, the event is generally passed along to the next input handler in the chain. One input handler that resides below Intuition's is the console device's handler. If your application's window has a console attached to it, the console device will generally convert events it receives into console code sequences, and send those to your console. In this manner, you can hear these events.

Because IntuiMessages and the IDCMP are the primary way in which applications receive communication from Intuition, discussions elsewhere in the manual frequently refer to events from Intuition as messages, IntuiMessages, or IDCMP messages. However, most of the information sent as IntuiMessages is also available through the console device, though that option is used less often. Elsewhere in this chapter, you can learn how getting your events through the console differs from getting them through your IDCMP.

Whichever way an application chooses to get its messages, it is frequently designed to be event-driven. That is to say, after some amount of initialization, the application will go into a state where it is waiting for some event to happen. This event could be an input event, or some other kind of event. Based on the event received, the application would take appropriate action, and return to its waiting state.

1.6 9 / Intuition Input / IDCMP Events and the Input Focus

Although at any given time many applications may be waiting for input, in most cases only the active application (the one with the currently active window) will receive IDCMP messages.

Since the IDCMP messages are, in general, directed to a single window, this window is said to have the input focus--the input from a variety of sources is focused on this single location.

The active window is generally selected by the user, although it is possible for applications to change the active window. See the "Intuition Windows" chapter for information on selecting or setting the active window. Be aware that changing the active window will change the input focus. Usually this change is performed following user action--the user selects a window with the mouse, or activates a new application. Changes to the input focus without user control, such as activating another window while the user is working in an application, may confuse the user. Perform such changes with great care.

Not all events are sent only to the active IDCMP. Some events, such as "disk inserted," may be useful to many programs, so Intuition translates these events into separate messages, one for each application.

1.7 9 Intuition Input and Output Methods / Intuition Output

Visual program output, the information written to the display, is sent through one of three channels.

* Imagery may be sent to the graphics library primitives. Graphics library includes functions for line drawing, area fill, specialized animation and output of text. See the graphics library chapters "Graphics Primitives", "Graphics Libraries and Text" and "Graphics Sprites, Bobs and Animation" for more on these functions.

* Use the Intuition library support functions for rendering text, graphical imagery, and line drawing. These provide some of the same functions as the graphics library routines, but the Intuition functions perform more of the detail work for you. See the chapter "Intuition Images, Line Drawing and Text" for more information on Intuition rendering functions. Also see, of course, the chapters on screens, windows, gadgets, menus and requesters for information on managing the display.

* Output character-based data via the console device. The console device is discussed in the next section.

1.8 9 Intuition Input and Output Methods / Console Device I/O

A program receives its input stream either directly from Intuition or via another mechanism known as the console device.

The console device may be used both as a source for input and as a

mechanism for output. Often, it is convenient to use only the console device for input and output. In particular, character-based programs can open the console and use it for all I/O without worrying about windows, bitmaps, or message ports.

The console device gives the program "cooked" input data, including key code conversions to ASCII and conversions of Intuition generated events, such as IDCMP_CLOSEWINDOW, to ANSI escape sequences.

The console device output provides features such as automatic line wrapping and scrolling. If an application just wants to output text, it may choose to use the console device, which provides formatted text with little fuss.

If the application is not character-based, it may be better for the it to use an IDCMP for input and render graphics and text directly through Intuition and the graphics library primitives.

If necessary, it is possible to open both the console device and an IDCMP for input. Such a program might need ASCII input, formatted output and the IDCMP verification functions (for example, to verify that it has finished writing to the window before the user can bring up a requester).

For more information on the console device, see the "Console Device" chapter of the Amiga ROM Kernel Reference Manual: Devices.

1.9 9 Intuition Input and Output Methods / Using the IDCMP

The IDCMP allow the application to receive information directly from Intuition. The program can use the IDCMP to learn about mouse, keyboard and other Intuition events. Also, certain useful Intuition features, most notably the verification functions (described under "IDCMP Flags" below), require that the IDCMP be opened, as this is the only mechanism available for accessing these features.

The IDCMP consists of a pair of message ports, which may be allocated and initialized by Intuition at the request of the program. Alternately, the application may choose to manage part of the allocation, such that one port is supplied by the application and one port is supplied by Intuition. These ports are standard Exec message ports, used to allow interprocess communications in the Amiga multitasking environment. To learn more about message ports and message passing, see the "Exec Messages and Ports" chapter.

The IDCMP is always associated with a window, it is not possible to have an IDCMP without an open window. The IDCMP is made up of several fields in the Window structure:

* IDCMPFlags stores the IDCMP flags currently set for this port. This field should never be directly set by the application; use the function ModifyIDCMP() or set them when the window is opened instead.

* UserPort is a pointer to the standard Exec message port where the application receives input event messages from Intuition

* WindowPort is a pointer to the reply message port used by Intuition. The messages sent by Intuition are set up such that ReplyMsg() will return them to this port.

To open these ports automatically, set at least one of the IDCMP flags in the OpenWindowTagList() call. To free these ports later in the program, call the function ModifyIDCMP() with NULL for the IDCMP flags or simply close the window.

Don't Reply Any Messages After the IDCMP is Freed.

If an IDCMP is freed, either by calling ModifyIDCMP() or by closing the window, Intuition will reclaim and deallocate all messages waiting at that port without waiting for a ReplyMsg(). If the program attempts to ReplyMsg() to an IntuiMessages after the IDCMP is closed, the system will probably crash.

If the IDCMP flags are NULL when the window is opened, no ports will be allocated when the window is created. To have Intuition allocate these ports later, call the function ModifyIDCMP() with any of the IDCMP flags set. (Starting in V37, ModifyIDCMP() returns NULL if it was unable to create the necessary message ports. Do not check the return code under V36 or earlier.)

Once the IDCMP is opened, with the ports allocated, the program can receive many types of information directly from Intuition, based on the IDCMP flags that are set.

The IDCMP allows the application to receive only the events that it considers important. The program can, for instance, choose to learn about gadget events but may not want to learn about other mouse or keyboard events. This is done by providing a "filter" or "mask" value for the IDCMP which tells Intuition which events it should send to this specific port. Only messages with a type matching one of the flags set in the Window structure's IDCMPFlags field will be sent to this port. These values may be set at creation time, or modified by calling the function ModifyIDCMP().

Messages sent to the IDCMP are instances of the structure IntuiMessage. This is an extended form of the Exec Message structure which allows Intuition to send user interface specific information to the application. The IntuiMessage structure is discussed at length below.

After the application opens an IDCMP, it must monitor the port for messages. At a minimum, this involves removing all messages from the port and replying to them. An event loop which processes messages arriving at the IDCMP is discussed below.

Standard IntuiMessage Event Loop Event Loop Example
Setting Up A Custom User Port IntuiMessages

1.10 9 / Using the IDCMP / Standard IntuiMessage Event Loop

The application should handle events quickly. Any delay in this handling will make the user interface appear sluggish to the user. Additionally,

certain events such as IDCMP_SIZEVERIFY may time-out if the application does not respond to them quickly (this is to help prevent system deadlocks). The action taken by Intuition when an event times-out may not match the action desired by the program. When IDCMP_SIZEVERIFY times out, the window sizing operation is cancelled by Intuition.

Code should be able to handle the case where there are multiple events waiting at the port. When events are being generated quickly, Intuition may post many events to the IDCMP before the application regains control. This can happen regardless of how fast the application processes the messages waiting at the port. Since messages queue up but signals do not, the application may not see a signal for each message posted. Because of these facts, the code should remove all the messages waiting at the port, regardless of the number, each time Wait() returns.

Code should also be able to handle the case where the signal is set but no events are waiting at the port. This could happen if a new message arrives at the IDCMP while an application is still processing the previous message. Since applications typically process all queued messages before returning to Wait(), the second message gets handled with the signal bit still set. The subsequent call to Wait() will return immediately even though no message is present. These cases should be quietly ignored.

1.11 9 / Using the IDCMP / Setting Up A Custom User Port

An application can use its own message port for the IDCMP instead of the one set up by Intuition, although some care is required.

As described earlier, IDCMP communication takes place through a pair of Exec message ports attached to a window: the UserPort and the WindowPort. The UserPort is the port where the application receives IDCMP messages from Intuition. The WindowPort is the reply port where Intuition receives replies from the application (via the ReplyMsg() function).

In the simplest case, Intuition allocates (and deallocates) both of these ports when the program opens a window with non-NULL IDCMP flags. Intuition will also allocate these ports if the application calls ModifyIDCMP() with non-NULL flags for a window that has NULL IDCMP flags. These port variables will be set to NULL if there is no message port allocated, otherwise they will contain a pointer to a message port.

If the WindowPort is not already opened when either OpenWindow() or ModifyIDCMP() is called, it will be allocated and initialized.

The UserPort is checked separately to see whether it is already opened.

When Intuition initializes the UserPort, it also allocates a signal bit with a call to AllocSignal(). Since the application makes the call to OpenWindowTagList() or ModifyIDCMP(), this signal bit is valid for the application's task. The address of the application's task is saved in the SigTask variable of the message port.

The program may choose to supply its own UserPort. This might be done in an environment where the program is using several windows and would prefer to monitor the input using only one message port. This is done by with

the following procedure:

1. Create a port for the IDCMP by calling either the Exec function CreateMsgPort() or the amiga.lib function CreatePort(), both of which return a pointer to a port. (CreateMsgPort() is a new Exec function in V36 and can therefore only be used on systems running Release 2 or a later version of the OS.)

2. Open the windows with no IDCMP flags set. This will prevent Intuition from allocating a port for this window.

3. Place a pointer to the port created in step 1 into the UserPort field of the Window structure.

4. Call ModifyIDCMP() to set the desired IDCMP flags for the port. Intuition will use the port supplied with the window.

   Be Careful with Shared IDCMP Ports.

   If the application is sharing an IDCMP among several windows, it must be very careful not to call ModifyIDCMP(window,NULL) for any of the windows that are using the shared port, as this will free the port and the signal bit.

5. When an application decides to close a window that has a shared IDCMP, there may be messages waiting at the port for any of the windows including the window being closed. It is essential that messages destined for a given window be removed and replied to before that window is closed.

   CloseWindowSafely(), listed in the next example, performs proper message cleanup before closing such a window. It also sets the window's UserPort to NULL so that Intuition knows not to delete the port, which should be done by the application in this case. It is incorrect (and dangerous) to simply call CloseWindow() on a window that has a shared IDCMP.

   Note that CloseWindowSafely() assumes that the window has a UserPort.

6. After all windows have been closed, and the port has been removed from each, delete the port that was created in step 1. Use the amiga.lib function DeletePort() (if CreatePort() was used) or the Exec function DeleteMsgPort() (if CreateMsgPort() was used).

Closing a Window with a Shared IDCMP

1.12 9 / Using the IDCMP / IntuiMessages

The IntuiMessage structure is an Exec Message that has been extended to include Intuition specific information. The ExecMessage field in the IntuiMessage is an actual instance of a Message structure and is used by Exec to manage the transmission of the message. The Intuition extensions of the IntuiMessage are used to transmit specialized Intuition data to the program.

```
    struct IntuiMessage
        {
        struct Message ExecMessage;
        ULONG Class;
        UWORD Code;
        UWORD Qualifier;
        APTR IAddress;
        WORD MouseX, MouseY;
        ULONG Seconds, Micros;
        struct Window *IDCMPWindow;
        struct IntuiMessage *SpecialLink;
        };
```

The IntuiMessage structure fields are as follows:

ExecMessage
: This field is maintained by Exec. It is used for linking the message into the system and broadcasting it to a message port. See the chapter "Exec Messages and Ports" for more information on the Message structure and its use.

Class
: Class contains the IDCMP type of this specific message. By comparing the Class field to the IDCMP flags, the application can determine the type of this message. Each message may only have a single IDCMP type.

Code
: Code contains data set by Intuition, such as menu numbers or special code values. The meaning of the Code field changes depending on the IDCMP type, or Class, of the message. Often the code field will simply contain a copy of the code of the input event which generated this IntuiMessage.

  For example, when the message is of class IDCMP_RAWKEY, Code contains the raw key code generated by the keyboard device. When the message is of class IDCMP_VANILLAKEY, Code contains the key mapped ASCII character.

Qualifier
: This contains a copy of the ie_Qualifier field that is transmitted to Intuition by the input device. This field is useful if your program handles raw key codes, since the Qualifier tells the program, for instance, whether or not the Shift key or Ctrl key is currently pressed. Check the <devices/inputevent.h> file for the definitions of the qualifier bits.

MouseX and MouseY
: Every IntuiMessage will have the mouse coordinates in these variables. The coordinates can either be expressed as absolute offsets from the upper left corner of the window, or expressed as the amount of change since the last reported positions (delta). If IDCMP_DELTAMOVE is set, then these numbers will represent delta positions from the last position. All messages will have zero in these values except IDCMP_MOUSEMOVE and IDCMP_MOUSEBUTTON events, which will have the correct delta values for the movement. If IDCMP_DELTAMOVE is not set, then these numbers are the actual window offset values.

Seconds and Micros
    These values are copies of the current system clock, in seconds and microseconds. They are set when Intuition generates the message.

    Microseconds (Micros) range from zero up to one million minus one. The 32 bits allocated to the Seconds variable has enough accuracy to count up to 139 years. Time is measured from Jan 1, 1978.

IAddress
    Typically this variable contains the address of some Intuition object, such as a gadget. The type of the object depends on the Class of the IntuiMessage. Do not assume that the object is of a certain type before checking the Class of the object.

    The IAddress pointer is defined only for the following IDCMP Classes. Do not attempt to dereference or otherwise interpret the IAddress field of any other type of IntuiMessage.

IntuiMessage Class	Meaning of IAddress Field
IDCMP_GADGETDOWN	IAddress points to the gadget.
IDCMP_GADGETUP	IAddress points to the gadget.
IDCMP_RAWKEY	IAddress points to the dead-key information.
IDCMP_IDCMPUPDATE	IAddress points to a tag item list.
Other classes	No meaning.

    In particular, for IDCMP_MOUSEMOVE IntuiMessages emanating from GACT_FOLLOWMOUSE gadgets, the IAddress field does not point to the gadget. Interpreting the IAddress as a gadget pointer and trying to access the gadget's fields before ascertaining that the event is an IDCMP_GADGETUP or IDCMP_GADGETDOWN event is incorrect, and can lead to subtle or serious problems.

    (Note that GadTools gadgets do arrange for the IAddress to point to the gadget when IDCMP_MOUSEMOVE messages appear).

IDCMPWindow
    Contains the address of the window to which this message was sent. If the application is sharing the window's UserPort between multiple windows, IDCMPWindow allows it to determine which of the windows the message was sent to.

SpecialLink
    For system use only.

1.13 9 Intuition Input and Output Methods / IDCMP Flags

The application specifies the information it wants Intuition to send to it via the IDCMP by setting IDCMP flags. These may be set either when opening the window or by calling ModifyIDCMP().

The flags set may be viewed as a filter, in that Intuition will only post IntuiMessages to an IDCMP if the matching flag is set. Thus, the application will only receive the IDCMP messages whose Class matches one

of the bits set in the window's IDCMP.

For many of these messages, there is a separation of the act of filtering these messages and causing Intuition to send the messages in the first place. For instance, menu help events may be activated for a window by setting the WA_MenuHelp attribute when the window is opened. However, the IDCMP will only receive the messages if the IDCMP_MENUHELP flag is set. If this flag is not set, then the events are passed downstream in the input and may be picked up by the console device.

Event Message Classes and Flags Verification Functions

1.14 9 / IDCMP Flags / Event Message Classes and Flags

Mouse Event Message Classes and Flags
Gadget Event Message Classes and Flags
Menu Event Message Classes and Flags
Requester Event Message Classes and Flags
Window Event Message Classes and Flags
Other Event Message Classes and Flags

1.15 9 / / Event Message Classes and Flags / Mouse Flags

IDCMP_MOUSEBUTTONS
Contains reports about mouse button up and down events. The events will be sent to the application only if they are not used internally by Intuition.

The Code field contains information on the specific mouse button event this message represents. The Code field will be equal to SELECTDOWN, SELECTUP, MENUDOWN, MENUUP, MIDDLEDOWN or MIDDLEUP, depending on the button pressed or released. In general, the select button is the left mouse button, the menu button is the right mouse button and the middle button is an optional third button usually located between the select and menu buttons.

Often, a mouse button event has extra meaning to Intuition, and the application may hear about it through a more specific message, for example a gadget or menu event. Other times, no event is generated at all, such as when the user depth-arranges a screen by clicking on the screen depth gadget. Note that menu button events are normally consumed by Intuition for menu handling. If an application wishes to hear IDCMP_MOUSEBUTTONS events for the menu button, it must set the WA_RMBTrap attribute for its window. See the "Intuition Windows" chapter for more information.

IDCMP_MOUSEMOVE
Reports about mouse movements, sent in the form of x and y coordinates relative to the upper left corner of the window. One message will be sent to the application for each "tick" of the mouse.

The application can opt to receive IDCMP_MOUSEMOVE events only while

certain gadgets are active, or during normal window operation. These events are sent whenever a gadget with GACT_FOLLOWMOUSE gadget is active, or for any window that has the WA_ReportMouse attribute set. This window attribute can be set or cleared by the application at will. See the "Intuition Windows" chapter for full details.

Requesting IDCMP_MOUSEMOVE messages can create a very large volume of messages arriving at the window's IDCMP. Do not request these messages unless the program is prepared to keep up with them. Starting in V36, Intuition limits the number of mouse move events that pile up at your IDCMP.

All IDCMP messages contain a mouse x and y position that can be absolute values or delta values. See IDCMP_DELTAMOVE, below. If the application requires a less frequent reporting of the mouse position, consider using IDCMP_INTUITICKS. While IDCMP_MOUSEMOVE events are generated by changes in the mouse's position, IDCMP_INTUITICKS IntuiMessages are based on a timer. Since they contain mouse coordinates, they effectively sample the mouse position. These message come often enough for many applications, but not so frequently as to swamp the application.

The program will not be sent IDCMP_MOUSEMOVE messages while Intuition has the layers of the screen locked (during menu operations and window sizing/dragging). This avoids problems of messages accumulating while the program is blocked, waiting to render into a locked layer.

IDCMP_DELTAMOVE

IDCMP_DELTAMOVE is not a message type, and events will never be sent to the application with this IDCMP identifier. This flag is a modifier, which changes how mouse movements are reported. When this flag is set, mouse movements are sent as delta values rather than as absolute positions.

The deltas are the amount of change of the mouse position from the last reported position. If the mouse does not move, then the delta values will be zero.

This flag works in conjunction with the IDCMP_MOUSEMOVE flag. When IDCMP_DELTAMOVE is set, IDCMP_MOUSEBUTTONS messages will also have relative values, instead of the absolute window position of the mouse.

Delta mouse movements are reported even after the Intuition pointer has reached the limits of the display. That is, if the pointer has reached the edge of the display and the user continues to move the mouse in the same direction, the IDCMP_MOUSEMOVE messages will continue to report changes in the mouse position even though the pointer is no longer moving.

1.16 9 / / Event Message Classes and Flags / Gadget Flags

IDCMP_GADGETDOWN

IDCMP_GADGETDOWN messages are sent when the user selects a gadget that was created with the GACT_IMMEDIATE flag set. The IntuiMessage

structure's IAddress field will contain a pointer to the selected gadget.

IDCMP_GADGETUP
IDCMP_GADGETUP messages are sent when the user selects a gadget that was created with the GACT_RELVERIFY flag set. The IntuiMessage structure's IAddress field will contain a pointer to the selected gadget.

IDCMP_CLOSEWINDOW
IDCMP_CLOSEWINDOW messages are sent when the user selects the window's close gadget. Intuition does not close the window when the close gadget is selected. Rather, an IDCMP_CLOSEWINDOW message is sent to the window's IDCMP. It is up to the application to clean up and close the window itself. If closing a window means losing some data (perhaps the spreadsheet the user was working on), it would be appropriate for the application to first confirm that the user really meant to close the window.

1.17 9 / / Event Message Classes and Flags / Menu Flags

IDCMP_MENUPICK
This flag indicates that the user has pressed the menu button. If a menu item was selected, the menu number of the menu item can be found in the Code field of the IntuiMessage. If no item was selected, the Code field will be equal to MENUNULL.

IDCMP_MENUVERIFY
This is a special verification mode which allows the program to confirm that it is prepared to handle Intuition rendering, in this case, allowing menus to be drawn in the screen.

This is a special kind of verification, in that any window in the entire screen that has this flag set must respond before the menu operations may proceed. Also, the active window of the screen is allowed to cancel the menu operation. This is unique to IDCMP_MENUVERIFY. Refer to the "Intuition Menus" for a complete description.

Also see the "Verification Functions" section below for more information.

IDCMP_MENUHELP
This message is sent by Intuition when the user selects the Help key while the menu system is activated. If a menu item was selected, the menu number of the menu item can be found in the Code field of the IntuiMessage. If no item was selected, the Code field will be equal to MENUNULL.

These messages will only be sent if the WA_MenuHelp attribute is set for the window.

The menu number returned in IDCMP_MENUHELP may specify a position that cannot be generated through normal menu activity. For instance, the menu number may indicate one of the menu headers with no item or

sub-item. See the chapter on "Intuition Menus" for more information.

1.18 9 / / Event Message Classes and Flags / Requester Flags

IDCMP_REQSET
Intuition sends an IDCMP_REQSET message to the window each time a requester opens in that window.

IDCMP_REQCLEAR
Intuition sends an IDCMP_REQCLEAR message to the window each time a requester is cleared from that window.

IDCMP_REQVERIFY
Set this flag to allow the application to ensure it is prepared for Intuition to render a requester in the window. With this flag set, Intuition sends the application a message that a requester is pending, and then waits for the application to reply before drawing the requester in the window.

If several requesters open in the window, Intuition asks the application to verify only the first one. After that, Intuition assumes that all output is being held off until all the requesters are gone.

By setting the IDCMP_REQSET and IDCMP_REQCLEAR flags, the application can track how many requesters are open in the window and when the last requester is cleared. Once all of the requesters are cleared from the window, it is safe to write to the window until another IDCMP_REQVERIFY is received.

See the "Verification Functions" section below for more discussion on using this flag.

1.19 9 / / Event Message Classes and Flags / Window Flags

IDCMP_NEWSIZE
Intuition sends this message after the user has resized the window. After receiving this, the program can examine the size variables in the window structure to discover the new size of the window. The message is sent, even if the size of the window did not actually change.

IDCMP_REFRESHWINDOW
This message is sent whenever the window needs refreshing. This flag makes sense only with windows that have a refresh type of WA_SimpleRefresh or WA_SmartRefresh.

As a minimum, the application must call BeginRefresh() and EndRefresh() for the window after receiving an IDCMP_REFRESHWINDOW event. Create the window with the WA_NoCareRefresh attribute if you do not want to manage these events. See the "Intuition Windows" chapter for details.

Most of the graphics library calls used for display output are compatible with Intuition, with the exception of ScrollRaster(). Intuition will not send an IDCMP_REFRESHWINDOW event when damage is caused to a window by ScrollRaster(). This may happen in a simple refresh window which is partially obscured by another window--the region that scrolls out from behind the front window will be damaged, but the window will receive no notification. Check the LAYERREFRESH bit in the Layer structure Flags field to see if damage did happen as a result of ScrollRaster().

IDCMP_SIZEVERIFY
Set this flag if the program must complete some operation before the user sizes the window. When the user sizes the window, Intuition sends an IDCMP_SIZEVERIFY message to the application and then waits until the program replies before allowing the user to size the window. See the "Verification Functions" section below for some things to consider when using this flag.

IDCMP_ACTIVEWINDOW and IDCMP_INACTIVEWINDOW
Set these flags to discover when the window becomes activated or deactivated.

1.20 9 / / Event Message Classes and Flags / Other Flags

IDCMP_VANILLAKEY
IDCMP_VANILLAKEY messages return keyboard events translated into the current default character keymap. The mapped character value is returned in the Code field of the IntuiMessage structure.

An IDCMP_VANILLAKEY message is sent only if the translation results in a single byte value, therefore the program cannot read the Help or function keys using IDCMP_VANILLAKEY.

Starting with V36, programs using IDCMP_VANILLAKEY which also require the additional information of special keys, such as the Help key and the function keys, may set both IDCMP_VANILLAKEY and IDCMP_RAWKEY. When this combination is used, all keypresses that map to single character values will be returned as IDCMP_VANILLAKEY events; all other keyboard events will be sent as IDCMP_RAWKEY messages. Note that IDCMP_VANILLAKEY processing uses all of the key-up events, so the application will only receive key-down events in the IDCMP_RAWKEY format.

IDCMP_RAWKEY
IDCMP_RAWKEY messages give the raw keycodes from the keyboard. The numeric value of the keycode is sent in the Code field. Separate codes are returned for key down and key up. Qualifier codes, such as Shift or Alt and whether this key is a repeat, may be found in the Qualifier field of the message.

In general, the application should not assume any correspondence between the keycode and the key value. Character positions on the keyboard change from country to country, and the application should respect the keymap set by the user.

Programs using IDCMP_RAWKEY messages should perform their own key mapping by calling the console.device function RawKeyConvert(), or the keymap.library function MapRawKey(). (The latter is a bit more convenient, but is only available under V36 and higher). The Autodoc for the MapRawKey() function shows how you can process so-called dead keys. A dead key is a key combination that has no immediate effect, but instead modifies a subsequent keystroke. For example, on the default keymap, Alt-F is a dead key for the acute accent mark. The sequence of Alt-F followed by the E key yields an é with an acute accent.

For an example of key mapping using the RawKeyConvert() call, see the rawkey.c example in the "Intuition Mouse and Keyboard" chapter.

The application can assume that certain keys will always return the same raw keycode, these keys do not have to be mapped. In general these keys are in the high part of the keymap, above hex 40, and includes all non-alphanumeric keys. The fixed keys include the function keys, backspace, delete, help and cursor keys.

IDCMP_NEWPREFS

IDCMP_NEWPREFS messages are sent when the system Preferences are changed by a call to SetPrefs(). The program can learn of these changes by setting this flag.

After receiving a message of class IDCMP_NEWPREFS, the application should call GetPrefs() to obtain a copy of the new Preferences.

Under the new Preferences scheme used in Release 2 and later versions of the OS, an IDCMP_NEWPREFS message will not always be sent when the user changes a Preferences setting. Only Preferences values available under V34, i.e., those that can be modified by a call to SetPrefs(), will cause an IDCMP_NEWPREFS message to be sent. New Preferences items such as overscan or font settings rely on filesystem notification for monitoring changes. See the chapter on "Preferences" for more information.

This message type is broadcast to all IDCMP that have this flag set, not just to the active window. If the application has this flag set, it should be prepared to handle the event even if it is not active.

IDCMP_DISKINSERTED and IDCMP_DISKREMOVED

When the user inserts or removes a floppy disk from any drive, Intuition will send one of these message types.

This message type is broadcast to all IDCMP that have this flag set, not just to the active window. If the application has this flag set, it should be prepared to handle the event even if it is not active.

IDCMP_INTUITICKS

Intuition sends these messages to the active window based on an internal timer which "ticks" roughly ten times a second. This provides the application with simple timer events from Intuition.

Intuition does not allow IDCMP_INTUITICKS events to accumulate at a port. After an IDCMP_INTUITICKS message has been sent to a port,

Intuition will not send another until the application replies to the first. This means that an application that has not been able to service the IDCMP for an extended period can expect at most one IDCMP_INTUITICKS message to be waiting at the port.

These events are to be used as "prods", and not as time counters. Do not rely on the timing accuracy of the event, or on the exact frequency at which they appear. Remember, IDCMP_INTUITICKS will only be sent to the active window. If the user selects another window, the events will no longer be received at the first window.

IDCMP_IDCMPUPDATE
Used for notification from Boopsi custom gadgets. See the chapter on "BOOPSI" for more information. The IAddress field contains a pointer to a tag item list. Tag lists are described in the chapter "Utility Library".

IDCMP_CHANGEWINDOW
This message provides the window with notification of any change in the size or position of a window.

There are two other message classes reserved for system use:

IDCMP_WBENCHMESSAGE
Special messages for Workbench, system use only.

IDCMP_LONELYMESSAGE
For internal tracking by Intuition, system use only.

1.21 9 / IDCMP Flags / Verification Functions

IDCMP_SIZEVERIFY, IDCMP_REQVERIFY and IDCMP_MENUVERIFY are exceptional in that Intuition sends an IntuiMessage to the application and then waits for the application to reply before Intuition proceeds. The application replies by calling the Exec function ReplyMsg().

The implication is that the user requested some operation but the operation will not happen immediately and, in fact, will not happen at all until the application says it is safe. Because this delay can be frustrating and intimidating, the program should strive to make the delay as short as possible. An application should always reply to a verification message as soon as possible.

These problems may be overcome by setting up a separate task to monitor the IDCMP and respond to incoming IntuiMessages immediately. This is recommended where there is heavy traffic through the IDCMP, which occurs when many IDCMP flags are set. Monitoring with a separate task may not be appropriate if the main program must synchronize with the event before it can respond to the message.

In previous versions of the operating system, it was not safe to leave any of the VERIFY functions enabled at a time when the task is unable to respond for a long period. This restriction included calls to AmigaDOS directly (with Open(), for example), or indirectly (with OpenLibrary(), for a disk based library, for example), when a VERIFY function was active.

This was because there are many cases where AmigaDOS will put up a requester prompting the user for input, and Intuition may end up waiting for the application to reply to the VERIFY message, while the application waits for the AmigaDOS call to finish. Prior to Release 2, this deadlock would freeze the Amiga.

Beginning with V36, Intuition will no longer wait forever for the application to respond to the verify messages. These messages will now time-out; that is, if the application does not respond within a set period, Intuition will act as if it had. Even in this case, though, the machine will appear to be locked up until the time-out occurs.

The application should use ModifyIDCMP() to turn off all VERIFY messages before calling AmigaDOS, or functions that may call AmigaDOS.

If the application sets up a separate task to monitor the IDCMP, and the task monitoring the IDCMP does not call AmigaDOS functions, and if the monitor task will always be able to reply to the VERIFY message without any help from the other task, then the above warning does not apply.

For additional information, see the IDCMP_MENUVERIFY discussion in the "Intuition Menus" chapter, the IDCMP_REQVERIFY discussion in the "Intuition Requesters and Alerts" chapter and the IDCMP_SIZEVERIFY discussion in the "Intuition Windows" chapter.

This message type is broadcast to all IDCMP on the screen that have this flag set, not just to the active window. If the application has this flag set, it should be prepared to handle the event even if it is not active.

1.22 9 Intuition Input and Output Methods / Function Reference

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

Table 9-1: Functions for Intuition Input and Output

Function	Description
ModifyIDCMP()	Change the message filter for an IDCMP. Starting in V37, this function has a return value.