Devices

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

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Devices

1.1 Amiga® RKM Devices: 6 Input Device

The input device is the central collection point for input events disseminated throughout the system. The best way to describe the input device is a manager of a stream with feeders. The input device itself and other modules such as the file system add events to the stream; so do input device "users" - programs or other devices that use parts of the stream or change it in some way. Feeders of the input device include the keyboard, timer and gameport devices. The keyboard, gameport, and timer devices are special cases in that the input device opens them and asks them for input. Users of the input device include Intuition and the console device.

NEW FEATURES FOR VERSION 2.0

Feature	Description
@{ "IECLASS_NEWPOINTERPOS " link 6-5-1}	Input Event Class
IECLASS_MENUHELP	Input Event Class
IECLASS_CHANGEWINDOW	Input Event Class
IESUBCLASS_COMPATIBLE	Input Event SubClass
IESUBCLASS_PIXEL	Input Event SubClass
IESUBCLASS_TABLET	Input Event SubClass
PeekQualifier()	Function

Compatibility Warning:

The new features for the 2.0 input device are not backwards compatible.

Additional Information on the Input Device

1.2 6 Input Device / Input Device Commands and Functions

Command	Operation
CMD_FLUSH	Purge all active and queued requests for the input device.
CMD_RESET	Reset the input port to its initialized state. All active and queued I/O requests will be aborted. Restarts the device if it has been stopped.
CMD_START	Restart the currently active input (if any) and resume queued I/O requests.
CMD_STOP	Stop any currently active input and prevent queued I/O requests from starting.
IND_ADDHANDLER	Add an input-stream handler into the handler chain.
IND_REMHANDLER	Remove an input-stream handler from the handler chain.
IND_SETMPORT	Set the controller port to which the mouse is connected.
IND_SETMTRIG	Set conditions that must be met by a mouse before a pending read request will be satisfied.
IND_SETMTYPE	Set the type of device at the mouse port.
IND_SETPERIOD	Set the period at which a repeating key repeats.
IND_SETTHRESH	Set the repeating key hold-down time before repeat starts.
IND_WRITEEVENT	Propagate an input event stream to all devices.

Input Device Function	
PeekQualifier()	Return the input device's current qualifiers. (V36)

Exec Functions as Used in This Chapter	
AbortIO()	Abort a command to the input device.
CheckIO()	Return the status of an I/O request.
CloseDevice()	Relinquish use of the input device.
DoIO()	Initiate a command and wait for completion (synchronous request).
OpenDevice()	Obtain shared use of the input device.
SendIO()	Initiate a command and return immediately (asynchronous

```
                request).

Exec Support Functions as Used in This Chapter
----------------------------------------------
CreateExtIO()    Create an extended I/O request structure of type
                 IOStdReq.  This structure will be used to communicate
                 commands to the input device.

CreatePort()     Create a signal message port for reply messages from the
                 input device.  Exec will signal a task when a message
                 arrives at the reply port.

DeleteExtIO()    Delete an I/O request structure created by CreateExtIO().

DeletePort()     Delete the message port created by CreatePort().
```

1.3 6 Input Device / Device Interface

```
The input device operates like the other Amiga devices.  To use it, you
must first open the input device, then send I/O requests to it, and then
close it when finished.  See the @{ "Introduction to Amiga System Devices" link  ↩
   ADCD_v1.2:Reference_Library/Devices/dev_1/1-3}
chapter for general information on device usage.

A number of structures are used by the input device to do its processing.
Some are used to pass commands and data to the device, some are used to
describe input events like mouse movements and key depressions, and one
structure is used to describe the environment for input event handlers.

The I/O request used by the input device for most commands is IOStdReq.

   struct IOStdReq
   {
       struct Message io_Message; /* message reply port */
       struct Device  *io_Device; /* device node pointer */
       struct Unit    *io_Unit;   /* unit */
       UWORD  io_Command;         /* input device command */
       UBYTE  io_Flags;           /* input device flags */
       BYTE   io_Error;           /* error code */
       ULONG  io_Length;          /* number of bytes to transfer */
       APTR   io_Data;            /* pointer to data area */
   };

See the include file exec/io.h for the complete structure definition.

Two of the input device commands - IND_SETTHRESH and IND_SETPERIOD -
require a time specification and must use a timerequest structure instead
of an IOStdReq.

   struct timerequest
   {
       struct IORequest tr_node;
       struct timeval tr_time;
   };
```

As you can see, the timerequest structure includes an IORequest structure. The io_Command field of theIORequest indicates the command to the input device and the timeval structure sets the time values. See the include file devices/timer.h for the complete structure definition.

In Case You Feel Like Reinventing the Wheel...

You could define a "super-IORequest" structure for the input device which would combine the IOStdReq fields with the timeval structure of the timerequest structure.

Opening The Input Device
Input Device Event Types
Closing The Input Device

1.4 6 / Device Interface / Opening The Input Device

Three primary steps are required to open the input device:

* Create a message port using CreatePort(). Reply messages from the device must be directed to a message port.

* Create an I/O request structure of type IOStdReq or timerequest. The I/O request created by the CreateExtIO() function will be used to pass commands and data to the input device.

* Open the Input device. Call OpenDevice(), passing the I/O request.

```
struct MsgPort  *InputMP;   /* Message port pointer */
struct IOStdReq *InputIO;   /* I/O request pointer */

if (InputMP=CreatePort(0,0) )
    if (InputIO=(struct IOStdReq *)
                CreateExtIO(InputMP,sizeof(struct IOStdReq)) )
        if (OpenDevice("input.device",0L,(struct IORequest *)InputIO,0))
            printf("input.device did not open\n");
```

The above code will work for all the input device commands except for the ones which require a time specification. For those, the code would look like this:

```
#include <devices/timer.h>

struct MsgPort  *InputMP;      /* Message port pointer */
struct timerequest *InputIO;   /* I/O request pointer */

if (InputMP=CreatePort(0,0) )
    if (InputIO=(struct timerequest *)
                CreateExtIO(InputMP,sizeof(struct timerequest)) )
        if (OpenDevice("input.device",0L,(struct IORequest *)InputIO,0))
            printf("input.device did not open\n");
```

1.5 6 / Device Interface / Input Device Event Types

The input device is automatically opened by the console device when the system boots. When the input device is opened, a task named "input.device" is started. The input device task communicates directly with the keyboard device to obtain raw key events. It also communicates with the gameport device to obtain mouse button and mouse movement events and with the timer device to obtain time events. In addition to these events, you can add your own input events to the input device, to be fed to the handler chain (see below).

The keyboard device is accessible directly (see the Keyboard Device chapter). However, once the input.device task has started, you should not read events from the keyboard device directly, since doing so will deprive the input device of the events and confuse key repeating.

The gameport device has two units. As you view the Amiga, looking at the gameport connectors, the left connector is assigned as the primary mouse input for Intuition and contributes gameport input events to the input event stream.

The right connector is handled by the other gameport unit and is currently unassigned. While the input device task is running, that task expects to read the input from the left connector. Direct use of the gameport device is covered in the Gameport Device chapter of this manual.

The timer device is used to generate time events for the input device. It is also used to control key repeat rate and key repeat threshold. The timer device is a shared-access device and is described in Timer Device chapter of this manual.

The device-specific commands are described below. First though, it may be helpful to consider the types of input events that the input device deals with. An input event is a data structure that describes the following:

* The class of the event-often describes the device that generated the event.
* The subclass of the event-space for more information if needed.
* The code-keycode if keyboard, button information if mouse, others.
* A qualifier such as "Alt key also down,"or "key repeat active".
* A position field that contains a data address or a mouse position count.
* A time stamp, to determine the sequence in which the events occurred.
* A link-field by which input events are linked together.
* The class, subclass, code and qualifier of the previous down key.

The full definitions for each field can be found in the include file devices/inputevent.h. You can find more information about input events in the Gameport Device and Console Device chapters of this manual.

The various types of input events are listed below.

Input Device Event Types

IECLASS_NULL	A NOP input event
IECLASS_RAWKEY	A raw keycode from the keyboard device
IECLASS_RAWMOUSE	The raw mouse report from the gameport device
IECLASS_EVENT	A private console event
IECLASS_POINTERPOS	A pointer position report
IECLASS_TIMER	A timer event
IECLASS_GADGETDOWN	Select button pressed down over a gadget (address in ie_EventAddress)
IECLASS_GADGETUP	Select button released over the same gadget (address in ie_EventAddress)
IECLASS_REQUESTER	Some requester activity has taken place.
IECLASS_MENULIST	This is a menu number transmission (menu number is in ie_Code)
IECLASS_CLOSEWINDOW	User has selected active window's Close Gadget
IECLASS_SIZEWINDOW	This window has a new size
IECLASS_REFRESHWINDOW	The window pointed to by ie_EventAddress needs to be refreshed
IECLASS_NEWPREFS	New preferences are available
IECLASS_DISKREMOVED	The disk has been removed
IECLASS_DISKINSERTED	The disk has been inserted
IECLASS_ACTIVEWINDOW	The window is about to be been made active
IECLASS_INACTIVEWINDOW	The window is about to be made inactive
IECLASS_NEWPOINTERPOS	Extended-function pointer position report (V36)
IECLASS_MENUHELP	Help key report during Menu session (V36)
IECLASS_CHANGEWINDOW	The Window has been modified with move, size, zoom, or change (V36)

There is a difference between simply receiving an input event from a device and actually becoming a handler of an input event stream. A handler is a routine that is passed an input event list. It is up to the handler to decide if it can process the input events. If the handler does not recognize an event, it leaves it undisturbed in the event list.

It All Flows Downhill.

Handlers can themselves generate new linked lists of events which can be passed down to lower priority handlers.

The InputEvent structure is used by the input device to describe an input event such as a keypress or a mouse movement.

```
struct InputEvent
{
    struct  InputEvent *ie_NextEvent;/* the next chronological event */
    UBYTE   ie_Class;           /* the input event class */
    UBYTE   ie_SubClass;        /* optional subclass of the class */
    UWORD   ie_Code;            /* the input event code */
    UWORD   ie_Qualifier;       /* qualifiers in effect for the event*/
    union
```

```
        {
            struct
            {
                WORD    ie_x;        /* the pointer position for event */
                WORD    ie_y;
            } ie_xy;
            APTR    ie_addr;         /* the event address */
            struct
            {
                UBYTE   ie_prev1DownCode;/* previous down keys for dead */
                UBYTE   ie_prev1DownQual;/* key translation: the ie_Code */
                UBYTE   ie_prev2DownCode;/* &low byte of ie_Qualifier for */
                UBYTE   ie_prev2DownQual;/* last & second last down keys */
            } ie_dead;
        } ie_position;
        struct timeval ie_TimeStamp;     /* the system tick at the event */
    };
```

The IEPointerPixel and IEPointerTablet structures are used to set the mouse position with the IECLASS_NEWPOINTERPOS input event class.

```
    struct IEPointerPixel
    {
        struct Screen       *iepp_Screen;   /* pointer to an open screen */
        struct
        {                           /* pixel coordinates in iepp_Screen */
            WORD    X;
            WORD    Y;
        } iepp_Position;
    };


    struct IEPointerTablet
    {
        struct
        {
            UWORD   X;
            UWORD   Y;
        } iept_Range;       /* 0 is min, these are max      */
        struct
        {
            UWORD   X;
            UWORD   Y;
        } iept_Value;       /* between 0 and iept_Range     */

        WORD iept_Pressure; /* -128 to 127 (unused, set to 0)  */
    };
```

See the include file devices/inputevent.h for the complete structure definitions.

For input device handler installation, the Interrupt structure is used.

```
struct Interrupt
{
    struct Node is_Node;
    APTR   is_Data;         /* server data segment */
    VOID   (*is_Code)();    /* server code entry   */
};
```

See the include file exec/interrupts.h for the complete structure definition.

1.6 6 / Device Interface / Closing The Input Device

Each OpenDevice() must eventually be matched by a call to CloseDevice(). All I/O requests must be complete before CloseDevice(). If any requests are still pending, abort them with AbortIO():

```
if (!(CheckIO(InputIO)))
    {
    AbortIO(InputIO);  /* Ask device to abort request, if pending */
    }
WaitIO(InputIO);       /* Wait for abort, then clean up */
CloseDevice((struct IORequest *)InputIO);
```

1.7 6 Input Device / Using the Mouse Port With the Input Device

To get mouse port information you must first set the current mouse port by passing an IOStdReq to the device with IND_SETMPORT
link ADCD_v1.2:Inc&AD2.1/autodocs/input/IND_SETMPORT} set in io_Command and a pointer to a byte set in io_Data. If the byte is set to 0 the left controller port will be used as the current mouse port; if it is set to 1, the right controller port will be used.

```
BYTE port = 1;      /* set mouse port to right controller */

InputIO->io_Data = &port;
InputIO->io_Flags = IOF_QUICK;
InputIO->io_Length = 1;
InputIO->io_Command = IND_SETMPORT;
BeginIO((struct IORequest *)InputIO);
if (InputIO->io_Error)
    printf("\nSETMPORT failed %d\n",InputIO->io_Error);
```

```
Put That Back!
--------------
The default mouse port is the left controller.  Don't forget to set
the mouse port back to the left controller before exiting if you
change it to the right controller during your application.
```

Setting The Conditions For A Mouse Port Report

1.8 6 / Setting The Conditions For A Mouse Port Report

You set the conditions for a mouse port report by passing an IOStdReq to the device with IND_SETMTRIG set in io_Command, the address of a GamePortTrigger structure set in io_Data and the length of the structure set in io_Length.

```
struct GamePortTrigger InputTR;

InputIO->io_Data = (APTR)InputTR;     /* set trigger conditions */
InputIO->io_Command = IND_SETMTRIG;   /* from InputTR */
InputIO->io_Length = sizeof(struct GamePortTrigger);
DoIO(InputIO);
```

The information needed for mouse port report setting is contained in a GamePortTrigger data structure which is defined in the include file devices/gameport.h.

```
struct GamePortTrigger
{
    UWORD    gpt_Keys;      /* key transition triggers */
    UWORD    gpt_Timeout;   /* time trigger (vertical blank units) */
    UWORD    gpt_XDelta;    /* X distance trigger */
    UWORD    gpt_YDelta;    /* Y distance trigger */
};
```

See the Gameport Device chapter of this manual for a full description of setting mouse port trigger conditions.

1.9 6 Input Device / Adding an Input Handler

You add an input-stream handler to the input chain by passing an IOStdReq to the device with IND_ADDHANDLER set in io_Command and a pointer to an Interrupt structure set in io_Data.

```
struct Interrupt *InputHandler;
struct IOStdReq  *InputIO

InputHandler->is_Code=ButtonSwap;         /* Address of code */
InputHandler->is_Data=NULL;               /* User Value passed in A1 */
InputHandler->is_Node.ln_Pri=100;         /* Priority in food chain */
InputHandler->is_Node.ln_Name=NameString; /* Name of handler */

InputIO->io_Data=(APTR)inputHandler;      /* Point to the structure */
InputIO->io_Command=IND_ADDHANDLER;       /* Set command ... */
DoIO((struct IORequest *)InputIO);        /* DoIO( ) the command */
```

Intuition is one of the input device handlers and normally distributes most of the input events.

Intuition inserts itself at priority position 50. The console device sits at priority position 0. You can choose the position in the chain at which your handler will be inserted by setting the priority field in the list-node part of the interrupt data structure you pass to this routine.

```
   Speed Saves.
   ------------
   Any processing time expended by a handler subtracts from the time
   available before the next event happens. Therefore, handlers for the
   input stream must be fast. For this reason it is recommended that the
   handlers be written in assembly.

 Rules For Input Device Handlers
 Removing An Input Handler
```

1.10 6 / Adding an Input Handler / Rules For Input Device Handlers

The following rules should be followed when you are designing an input handler:

* If an input handler is capable of processing a specific kind of an input event and that event has no links (ie_NextEvent = 0), the handler can end the handler chain by returning a NULL (0) value.

* If there are multiple events linked together, the handler is free to unlink an event from the input event chain, thereby passing a shorter list of events to subsequent handlers. The starting address of the modified list is the return value.

* If a handler wishes to add new events to the chain that it passes to a lower-priority handler, it may initialize memory to contain the new event or event chain. The handler, when it again gets control on the next round of event handling, should assume nothing about the current contents of the memory blocks attached to the event chain. Lower priority handlers may have modified the memory as they handled their part of the event. The handler that allocates the memory for this purpose should keep track of the starting address and the size of this memory chunk so that the memory can be returned to the free memory list when it is no longer needed.

Your assembly language handler routine should be structured similar to the following pseudo-language statement:

```
   newEventChain = yourHandlerCode(oldEventChain, yourHandlerData);
        d0       =                  a0              a1
```

where:

* yourHandlerCode is the entry point to your routine.

* oldEventChain is the starting address for the current chain of input events.

* yourHandlerData is a user-definable value, usually a pointer to some data structure your handler requires.

* newEventChain is the starting address of an event chain which you are passing to the next handler, if any.

When your handler code is called, the event chain is passed in A0 and the handler data is passed in A1. (You may choose not to use A1.) When your code returns, it should return the pointer to the event chain in D0. If all of the events were removed by the routine, return NULL. A NULL (0) value terminates the handling thus freeing more CPU resources.

Memory that you use to describe a new input event that you have added to the event chain is available for reuse or deallocation when the handler is called again or after the IND_REMHANDLER command for the handler is complete. There is no guarantee that any field in the event is unchanged since a handler may change any field of an event that comes through the food chain.

```
Do Not Confuse the Device.
--------------------------
Altering a repeat key report will confuse the input device when it
tries to stop the repeating after the key is raised under pre-V36
Kickstart.
```

Because IND_ADDHANDLER installs a handler in any position in the handler chain, it can, for example, ignore specific types of input events as well as act upon and modify existing streams of input. It can even create new input events for Intuition or other programs to interpret.

1.11 6 / Adding an Input Handler / Removing An Input Handler

You remove a handler from the handler chain by passing an IOStdReq to the device IND_REMHANDLER set in io_Command and a pointer to the Interrupt structure used to add the handler.

```
struct Interrupt *InputHandler;
struct IOStdReq  *InputIO;

InputIO->io_Data=(APTR)InputHandler;   /* Which handler to REM */
InputIO->io_Command=IND_REMHANDLER;    /* The REM command */
DoIO((struct IORequest *)InputIO);     /* Send the command */
```

1.12 6 Input Device / Writing Events to the Input Device Stream

Typically, input events are internally generated by the timer device, keyboard device, and input device.

An application can also generate an input event by setting the appropriate fields for the event in an InputEvent structure and sending it to the input device. It will then be treated as any other event and passed through to the input handler chain. However, I/O requests for IND_WRITEVENT cannot be made from interrupt code.

You generate an input event by passing an IOStdReq to the device with IND_WRITEEVENT set in io_Command, a pointer to an InputEvent structure set in io_Data and the length of the structure set in io_Length.

```
   struct InputEvent *FakeEvent;
   struct IOStdReq   *InputIO;

   InputIO->io_Data=(APTR)FakeEvent;
   InputIO->io_Length=sizeof(struct InputEvent);
   InputIO->io_Command=IND_WRITEEVENT;
   DoIO((struct IORequest *)InputIO);

   You Know What Happens When You Assume.
   --------------------------------------
   This command propagates the input event through the handler chain.
   The handlers may link other events onto the end of this event or
   modify the contents of the data structure you constructed in any way
   they wish. Therefore, do not assume any of the data will be the same
   from event to event.

 Setting The Position Of The Mouse
```

1.13 6 / Writing Events to Input Device Stream / Setting Position Of Mouse

One use of writing input events to the input device is to set the position of the mouse pointer. The mouse pointer can be positioned by using the input classes IECLASS_POINTERPOS and @{ "IECLASS_NEWPOINTERPOS " link ADCD_v1.2:Inc&AD2.1/includes/devices/inputevent.h/main}.

There are two ways to set the position of the mouse pointer using the pre-V36 Kickstart input class IECLASS_POINTERPOS:

* At an absolute position on the current screen.
* At a position relative to the current mouse pointer position on the current screen.

In both cases, you set the Class field of the InputEvent structure to IECLASS_POINTERPOS, ie_X with the new x-coordinate and ie_Y with the new y-coordinate. Absolute positioning is done by setting ie_Qualifier to NULL and relative positioning is done by setting ie_Qualifier to RELATIVE_MOUSE.

Once the proper values are set, pass an IOStdReq to the input device with a pointer to the InputEvent structure set in io_Data and io_Command set to IND_WRITEEVENT.

There are three ways to set the mouse pointer position using IECLASS_NEWPOINTERPOS:

* At an absolute x-y coordinate on a screen-you specify the exact location of the pointer and which screen.
* At an relative x-y coordinate-you specify where it will go in relation to the current pointer position and which screen.
* At a normalized position on a tablet device-you specify the maximum x-value and y-value of the tablet and an x-y coordinate between them and the input device will normalize it to fit.

The basic steps required are the same for all three methods.

* Get a pointer to the screen where you want to position the pointer. This is not necessary for the tablet device.

* Set up a structure to indicate the new position of the pointer.

For absolute and relative positioning, you set up an IEPointerPixel structure with iepp_Position.X set to the new x-coordinate, iepp_Position.Y set to the new y-coordinate and iepp_Screen set to the screen pointer. You set up an InputEvent structure with ie_SubClass set to IESUBCLASS_PIXEL, a pointer to the IEPointerPixel structure set in ie_EventAddress, IECLASS_NEWPOINTERPOS set in Class, and ie_Qualifier set to either IEQUALIFIER_RELATIVEMOUSE for relative positioning or NULL for absolute positioning.

For tablet positioning, you set up an IEPointerTablet structure with iept_Range.X set to the maximum x-coordinate and iept_Range.Y set to the maximum y-coordinate, and iept_Value.X set to the new x-coordinate and iept_Value.Y set to the new y-coordinate. You set up an InputEvent structure with a pointer to the IEPointerTablet structure set in ie_EventAddress, ie_SubClass to IESUBCLASS_TABLET and Class set to IECLASS_NEWPOINTERPOS.

Finally, for all three methods, pass an IOStdReq to the device with a pointer to the InputEvent structure set in io_Data and io_Command set to IND_WRITEEVENT.

The following example sets the mouse pointer at an absolute position on a public screen using IECLASS_NEWPOINTERPOS. Notice that it uses V36 functions wherever possible.

Set_Mouse.c

1.14 6 Input Device / Setting the Key Repeat Threshold

The key repeat threshold is the number of seconds and microseconds a user must hold down a key before it begins to repeat. This delay is normally set by the Preferences tool or by Intuition when it notices that the Preferences have been changed, but you can also do it directly through the input device.

You set the key repeat threshold by passing a timerequest with IND_SETTHRESH set in io_Command and the number of seconds to delay set in tv_secs and the number of microseconds to delay set in tv_micro.

```
#include <devices/timer.h>

struct timerequest *InputTime;/* Init with CreateExtIO() before using */

InputTime->tr_time.tv_secs=1;         /* 1 second */
InputTime->tr_time.tv_micro=500000;   /* 500000 microseconds */
InputTime->tr_node.io_Command=IND_SETTHRESH;
DoIO((struct IORequest *)InputTime);
```

```
The code above will set the key repeat threshold to 1.5 seconds.
```

1.15 6 Input Device / Setting the Key Repeat Interval

```
The key repeat interval is the time period, in seconds and microseconds,
between key repeat events once the initial key repeat threshold has
elapsed. (See Setting the Key Repeat Threshold.)  Like the key
repeat threshold, this is normally issued by Intuition and preset by the
Preferences tool.

You set the key repeat interval by passing a timerequest with
IND_SETPERIOD set in io_Command and the number of seconds set in tv_secs
and the number of microseconds set in tv_micro. struct timerequest
*InputTime; /* Initialize with CreateExtIO() before using */

   InputTime->tr_time.tv_secs=0;
   InputTime->tr_time.tv_micro=12000;  /* .012 seconds */
   InputTime->tr_node.io_Command=IND_SETPERIOD;
   DoIO((struct IORequest *)InputTime);

The code above sets the key repeat interval to .012 seconds.

   The Right Tool For The Right Job.
   ---------------------------------
   As previously stated, you must use a timerequest structure with
   IND_SETTHRESH and IND_SETPERIOD
input/IND_SETPERIOD}.
```

1.16 6 Input Device / Determining the Current Qualifiers

```
Some applications need to know whether the user is holding down a
qualifier key or a mouse button during an operation.  To determine the
current qualifiers, you call the input device function PeekQualifier().

PeekQualifier() returns what the input device considers to be the current
qualifiers at the time PeekQualifier() is called (e.g., keyboard
qualifiers and mouse buttons).   This does not include any qualifiers
which have been added, removed or otherwise modified by input handlers.

In order to call the function, you must set a pointer to the input device
base address.  The pointer must be declared in the global data area of
your program.  Once you set the pointer, you can call the function. You
must open the device in order to access the device base address.

PeekQualifier() returns an unsigned word with bits set according to the
qualifiers in effect at the time the function is called.  It takes no
parameters.

   struct Library *InputBase;      /* Input device base address pointer */

   VOID main(VOID)
```

```
   {
   struct IOStdReq   *InputIO;     /* I/O request block */
   UWORD  Quals;                   /* qualifiers */
     .
     .
     .
   if (!OpenDevice("input.device",NULL,(struct IORequest *)InputIO,NULL))
        {
        /* Set input device base address in InputBase */
        InputBase = (struct Library *)InputIO->io_Device;

        /* Call the function */
        Quals = PeekQualifier();
        .
        .
        .
        CloseDevice(InputIO);
        }
   }
```

The qualifiers returned are listed in the table below.

Bit	Qualifier	Key or Button
0	IEQUALIFIER_LSHIFT	Left Shift
1	IEQUALIFIER_RSHIFT	Right Shift
2	IEQUALIFIER_CAPSLOCK	Caps Lock
3	IEQUALIFIER_CONTROL	Control
4	IEQUALIFIER_LALT	Left Alt
5	IEQUALIFIER_RALT	Right Alt
6	IEQUALIFIER_LCOMMAND	Left-Amiga
7	IEQUALIFIER_RCOMMAND	Right-Amiga
12	IEQUALIFIER_MIDBUTTON	Middle Mouse
13	IEQUALIFIER_RBUTTON	Right Mouse
14	IEQUALIFIER_LEFTBUTTON	Left Mouse

1.17 6 Input Device / Input Device and Intuition

There are several ways to receive information from the various devices that are part of the input device. The first way is to communicate directly with the device. This method is not recommended while the input device task is running - which is most of the time. The second way is to become a handler for the stream of events which the input device produces. That method is shown above.

The third method of getting input from the input device is to retrieve the data from the console device or from the IDCMP (Intuition Direct Communications Message Port). These are the preferred methods for applications in a multitasking environment because each application can receive juts its own input (i.e., only the input which occurs when one of its window is active). See the "Intuition" chapters of Amiga ROM Kernel Reference Manual: Libraries for more information on IDCMP messages. See the Console Device chapter of this manual for more information on console device I/O.

1.18 6 Input Device / Additional Information on the Input Device

Additional programming information on the input device can be found in
the include files and the autodocs for the input device. Both are
contained in the Amiga ROM Kernel Reference Manual: Includes and Autodocs.

Input Device Information

INCLUDES	devices/input.h
	devices/input.i
	devices/inputevent.h
	devices/inputevent.i
AUTODOCS	input.doc