Chapter 15
Miscellaneous Tools

The MiscTools have been modified to take Discovery and Universe into account. Two new types of calls have been added to make developing easier, first a queue manager has been added that gives you a tool for general purpose queue management, and Interupt state calls have been added to give you more information about when an interupt has occured and what state the machine was in when the interupt occured.

Also added is an alternative entry into the code that reads the mouse location that does not journal, and a routine to return the address of Resource Converter for code resources.

Changes

The ReadBParam and WriteBParam have been modified to reflect an internal change in the battery RAM structure for the Discovery ROM's. Now, instead of the number that you are passing being the actual batteryRAM address to load and store, it is simply an ID of the parameter you want to use. The existing ID's you currently are using will still be valid, and the calls will function for you as they always did.

We have modified the mouse calls in misc tools and the event manager in the Discovery ROM's so that they will no longer require you to have the slot 4 firmware enabled to read the mouse. The net result of this being that users will get an "extra" slot that they can use for peripheral cards.

We have changed misc tools to allow greater compatability with journaling. The readmouse call will now be journaled, its journalcode is 6 (see the event manager chapter for more info). In addition to that we have added a new call ReadMouse2, which does not support journaling. Applications should always call ReadMouse in order to support DA's which might take advantage of journaling.

We have added new vectors to the system, so we have changed the SetVector and GetVector calls to include these new vectors. In addition to this, we have removed the error checking from the call, so if you pass the SetVector call a bad vector ID you will probably trash someplace in memory that you did not want to trash. The added vectors are:

$80	Vector to Memory Mover
$81	Vector to Set System Speed
$82	Vector to Slot Arbiter
$86	Hardware Independent interrupt vector
$87	IRQ.MIDI (Midi interrupt vector)

Queue Handling

The MiscTools now provide a general Queue handling capability for other tools and applications. These calls deal with queues of a standard format, each queue entry must have a standard header, that will be followed by caller defined data of any length. The Queue handling routines will take care of adding and remove individual entries from the queue. The format of the standard header is:

Reserved	LONG	Link to next task in queue set up by queue handler
Reserved	WORD	Used by individual routines.
Signature	WORD $A55A	Signature (like heartbeat task header)
MoreData		The rest is defined by the individual queue handler.

Both the add and remove routines check to see that the signature word is valid and return an error if they are not.

Each of the queue handling calls requires a pointer to the first entry in each queue (HeaderPtr). You must create this first entry and maintain the pointer to it yourself.

AddToQueue $2E03

Input
- NewEntryPtr LONG
- HeaderPtr LONG

Output
- None

It adds the indicated entry(NewEntryPtr) to the Queue indicated by HeaderPtr. If the entry is already in the queue (list), the AlreadyInQueue error is returned. If the tag word is invalid, the invalidTag error is returned.

Possible Errors:

InvalidTag = $0381
AlreadyInQueue = $0382

DeleteFromQueue $2F03

Input
- EntryPtr LONG
- HeaderPtr LONG

Output
- None

Removes the indicated entry (EntryPtr) from the Queue indicated by HeaderPtr. If the entry cannot be found in the queue, the NotInList error is returned. If the tag word is invalid, the InvalidTag error is returned.

Possible Errors:

NotInList = $0380
InvalidTag = $0381

Interrupt State Calls

When writing interupt based debuggers or interupt handlers that must know the state of the program that was running when the interupt occured. We've added three calls to the Misc Tools to help support this, GetInterruptState, SetInterruptState and GetIntStateRecSize, these calls all deal with the system interupt variables. GetInterruptState copies variables from system memory into a specified record, SetInterruptState copies variables from a specified record into system memory and GetIntStateRecSize returns the size (in bytes) of the interrupt state record.

The calls are designed so that we can change the amount of information that we save and restore during interrupts and debuggers can still work. The debugger would only ask for and set the amount of information it can deal with.

The current interrupt state record is:

IntStateRecord	
irq_A	word
irq_X	word
irq_Y	word
irq_S	word
irq_D	word
irq_P	byte
irq_DB	byte
irq_e	byte
irq_K	byte
irq_PC	word
irq_state	byte
irq_shadow	word
irq_mslot	byte

The current interrupt state record size is 20 bytes.

SetInterruptState — Call $3003

Input

IntStateRecPtr	LONG
BytesDesired	WORD

Output

None

Copies the number of indicated bytes from the specified state record into the system interupt variables.

GetInterruptState — Call $3103

Input

IntStateRecPtr	LONG
BytesDesired	WORD

Output

None

Copies the number of indicated bytes into the specified state record from the system interupt variables.

GetIntStateRecSize Call $3203

Input
space for integer WORD

Output
SizeOfRecord integer

Returns the size (in bytes) of the interrupt state record.

Other New Calls

ReadMouse2 $3303

Input
space WORD
space WORD
space WORD

Output
xPosition WORD
yPosition WORD
Status/Mode WORD

This call is identical to ReadMouse in every way except that it does not support journaling. Applications should never make this call.

Possible Errors:

$0309 unCnctdDevErr - pointing device is not connected

GetCodeResConverter $3403

Input
space Long

Output
PtrToCodeResConverter LONG

This call returns the address of a routine that can be used to load code resources. It is part of miscellaneous tools because the loader is not in directly accessable memory. (It lives in the bank 1 language card which may or may not be switched in at any given time.)

You would use this call in conjunction with the resource manager's ResourceConverter call. For example, the Control Manager makes the following call when it is started up:

```
ResourceConverter(GetCodeResConverter,rCtlDefProc.LogConverterIn+SysConverterList)
```

All future calls to the resource manager that load resources of type rCtlDefProc will use the routine in the Misc Tools to bring the resource into memory.

Change History

6 Jan 89 Mensch

Add ReadMouse2 call

13 Jan 89 Steven Glass

Fixed the inputs to AddToQueue and DeleteFromQueue; they were backwards on the stack.
Added possible error to AddToQueue (AlreadyInList).
Added GetCodeResConverter call.
Fixed call numbers of GetInterruptState and SetInterruptState.