Cortland Miscellaneous Tools

July 16, 1986

Written by Ray Montagne & Eagle Berns

Revision History.....

March 10, 1986	Ver. 0.81	R. Montagne	Major revisions to the Miscellaneous Tool Set have occured. The Integer Math functions have been removed, and now comprise the INTEGER MATH TOOL SET. The Pascal and Basic I/O functions have also been removed, and are now found in the TEXT TOOL SET. A stack pointer indicator (sp—>) has been added to the parameter lists for clarity. Basic functionality of most tool functions remaining in the MISCELLANEOUS TOOL SET has not changed. However, all of the function numbers have changed. Many of the functional descriptions have been rewritten for clarity. Functions that have changed are: (1) INTERRUPT CONTROL TOOLS (2) FIRMWARE FLAG TOOLS (3) INTERRUPT ENABLE STATUS TOOLS
March 12, 1986	Ver. 0.82	R. Montagne	System Death Manager. Additional information on interrupts source control (Keyboard interrupts). Additional information on the environment when using Firmware Entry. Additional information on installing ROM based tasks into the HeartBeat queue.
April 18, 1986	Ver. 0.83	R. Montagne	ID Manager Type 8. System Death Error Codes. Additional Vectors.
April 23, 1986	Ver. 0.84	R. Montagne	Added vectors for STEP and TRACE. Additional parameters in the GET ADDRESS function. This is the BETA 2.0 Implementation.
April 29, 1986	Ver. 0.85	R. Montagne	Added functions to set and get clamps for absolute devices. Mouse calls will return an error if the card is not switched in rather than call the system death manager.
May 9, 1986	Ver. 0.86	R. Montagne	ID Manager ID assignments. Read ASCII time (state of MSB). Set/Get Vectors.
June 11, 1986	Ver. 0.87	R. Montagne	Added SETUP FILE ID to ID Manager.
June 26, 1986	Ver. 0.88	R. Montagne	No functional change, just added examples.
July 9, 1986	Ver. 0.89	Montagne/Berns	Added SCRAP MANAGER ID tag. Eagle's Examples!! System Death Syntax. System Death messages and Language Card. More descriptive death codes. Stack example in MUNGER was corrected.
July 16, 1986	Ver. 0.90	Montagne	Corrected tool number in death codes.

Miscellaneous Tools. So far the tools we have specified fall into broad catagories and each deserve their own tool set. Unfortunately, ther are a number of routines in the firmware that do not fall into any of these categories but still must be accessed from native mode. These routines include:

APPLE][entry points	Mouse support	Clock support
Battery ram support	Interrupt support	ID Tag managment
VBL or HeartBeat managment	System Death managment	

Standard Tool Set Calls.

MTBootInit Function number = $01

This tool call clears the TickCounter and the HeartBeat task link pointer. It also sets the Mouse flag to 'NOT FOUND'. A block of memory with a length of NIL is requested from the memory manager for use by the ID tag manager.

Example:

```
_MTBOOTINIT
```

MTStartUp Function number = $02

This does nothing.

Example:

```
_MTSTARTUP
```

MTShutDown Function number = $03

This does nothing.

Example:

```
_MTSHUTDOWN
```

MTVersion Function number = $04

Input	Word	Space for result	
sp—>			
Output	Word	Version number	
sp—>			

This tool returns the version number of the Miscellaneous Tool Set.

Example:

```
        PEA     $0000       ; SPACE FOR RESULT
        _MTBOOTINIT
```

MTReset Function number = $05

This tool call clears the HeartBeat queue link pointer and sets the Mouse flag to 'NOT FOUND'.

Example:

```
_MTRESET
```

MTStatus Function number = $06

Input	Word	Space for result	
sp—>			
Output	Word	Status ($0000=Inactive, $FFFF=Active)	
sp—>			

This tool returns a status that indicates that the Miscellaneous Tool Set is active.

Example:

```
PEA             $0000           ; SPACE FOR RESULT
_MTSTATUS
```

MTSpare1 Function number = $07

This does nothing.

Example:

```
_MTSPARE1
```

MTSpare2 Function number = $08

This does nothing.

Example:

```
_MTSPARE2
```

Battery Ram Tools. These routines allow the non volatile battery backed up ram to be read or written.

WriteBRam Function number = $09

	Input	LongWord	Buffer Address
sp—>			

The 252 bytes of data at the memory location specified by the **Buffer Address** plus four bytes of checksum data is written to the battery ram.

Example:

```
PUSHLONG   #LABEL        ; BUFFER ADDRESS
_WRITEBRAM
```

ReadBRam Function number = $0A

	Input	LongWord	Buffer Address
sp—>			

The 252 bytes of data plus four bytes of checksum data is read from the battery ram and stored at the memory location specified by the **Buffer Address.**

Example:

```
PUSHLONG   #LABEL        ; BUFFER ADDRESS
_READBRAM
```

WriteBParam Function number = $0B

	Input	Word	Data (low byte only)
	Input	Word	Parameter Reference Number (0-255)
sp—>			

Data is written to the battery ram location specified by the **Parameter Reference Number.**

Example:

```
PEA        $0005         ; DATA IN LOW BYTE
PEA        $0028         ; REF = STARTUP SLOT
_WRITEBPARAM
```

ReadBParam Function number = $0C

	Input	Word	Space for result
	Input	Word	Parameter Reference Number (0-255)
sp—>			
	Output	Word	Data (low byte only)
sp—>			

Example:

```
PEA     $0000     ; SPACE FOR RESULT
PEA     $0028     ; REF = STARTUP SLOT
_READBPARAM
```

Data is read from the battery ram location specified by the **Parameter Reference Number.**

Battery Ram Parameter Reference Numbers:

$00	Port 1 Printer / Modem
$01	Port 1 Line Length
$02	Port 1 Delete line feed after carriage return
$03	Port 1 Add line feed after carriage return
$04	Port 1 Echo
$05	Port 1 Buffer
$06	Port 1 Baud
$07	Port 1 Data / Stop Bits
$08	Port 1 Parity
$09	Port 1 DCD Handshake
$0A	Port 1 DSR Handshake
$0B	Port 1 Xon / Xoff Handshake
$0C	Port 2 Printer / Modem
$0D	Port 2 Line Length
$0E	Port 2 Delete line feed after carriage return
$0F	Port 2 Add line feed after carriage return
$10	Port 2 Echo
$11	Port 2 Buffer
$12	Port 2 Baud
$13	Port 2 Data / Stop Bits
$14	Port 2 Parity
$15	Port 2 DCD Handshake
$16	Port 2 DSR Handshake
$17	Port 2 Xon / Xoff Handshake
$18	Display Color / Monochrome
$19	Display 40 / 80 column
$1A	Display Text Color
$1B	Display Background Color
$1C	Display Border Color
$1D	50 / 60 Hertz
$1E	User Volume
$1F	Bell Volume
$20	System Speed
$21	Slot 1 Internal / External
$22	Slot 2 Internal / External
$23	Slot 3 Internal / External
$24	Slot 4 Internal / External
$25	Slot 5 Internal / External
$26	Slot 6 Internal / External
$27	Slot 7 Internal / External
$28	Startup Slot
$29	Text Display Language
$2A	Keyboard Language
$2B	Keyboard Buffering
$2C	Keyboard Repeat Speed
$2D	Keyboard Repeat Delay
$2E	Double Click Time

$2F	Flash Rate
$30	Shift Caps / Lower Case
$31	Fast Space / Delete Keys
$32	Dual Speed
$33	High Mouse Resolution
$34	Month / Day / Year Format
$35	24 Hour / AM-PM Format
$36	Minimum Ram for RAMDISK
$37	Maximum Ram for RAMDISK
$38-40	Count / Languages
$41-51	Count / Layouts
$52-7F	Reserved
$80	AppleTalk Node Number
$81-A1	Operating system variables
$A2-FB	Reserved
$FC-FF	Checksum

Clock Tools. These routines allow the clock to be set or read. Setting the clock requires that the time be passed as an input paramter in a hex format. Two tools are provided for reading the clocl. One returns time in a hex format, while the other returns time in an ASCII format.

ReadTimeHex Function number = $0D

Input	Word	Space for result	
Input	Word	Space for result	
Input	Word	Space for result	
Input	Word	Space for result	
sp—>			
Output	Byte	Day of Week	(0-6 where 0 = Sunday)
Output	Byte	null	
Output	Byte	Month	(0-11 where 0 = January)
Output	Byte	Day	(0-30)
Output	Byte	Current Year minus 1900	
Output	Byte	Hour	(0-23)
Output	Byte	Minute	(0-59)
Output	Byte	Second	(0-59)
sp—>			

Returns current time in Hex format.

Example:

```
PEA         $0000       ; SPACE FOR RESULT
PEA         $0000       ; SPACE FOR RESULT
PEA         $0000       ; SPACE FOR RESULT
PEA         $0000       ; SPACE FOR RESULT
_READTIMEHEX
```

WriteTimeHex Function number = $0E

Input	Byte	Month	(0-11 where 0 = January)
Input	Byte	Day	(0-30)
Input	Byte	Current Year minus 1900	
Input	Byte	Hour	(0-23)
Input	Byte	Minute	(0-59)
Input	Byte	Second	(0-59)
sp—>			

Sets the current time using Hex format.

Example:

```
PEA         $0105       ; FEBRUARY, 5TH
PEA         $560A       ; 1986, 10TH HOUR
PEA         $1900       ; 25TH MINUTE, 0 SEC.
_WRITETIMEHEX
```

ReadAsciiTime Function number = $0F

Input LongWord ASCII buffer address
sp—>

Reads elapsed time since January 1, 00:00:00 1904, and converts to ASCII time output which is placed in the applications buffer. Note that ASCII time always outputs twenty characters with the MSB of each character set to a one. ASCII time format is defined by the format set up in the battery ram by the control panel. Format versus the battery ram parameter value is shown below:

Date Format	Time Format	ASCII Time Format
0	0	mm/dd/yy HH:MM:SS AM or PM
1	0	dd/mm/yy HH:MM:SS AM or PM
2	0	yy/mm/dd HH:MM:SS AM or PM
0	1	mm/dd/yy HH:MM:SS
1	1	dd/mm/yy HH:MM:SS
2	1	yy/mm/dd HH:MM:SS

Where: HH = Hour
MM = Minute
SS = Second
mm = Month
dd = Day
yy = Year

Example:

```
PUSHLONG    #LABEL          ; BUFFER ADDRESS
_READASCIITIME
```

Vector Initialization Tools. These tools allow the application to set or get the current vector for the interrupt handlers.

SetVector Function number = $10

	Input	Word	Vector Reference Number
	Input	LongWord	Address
sp—>			

Sets the vector address for the interrupt manager or handler specified by the vecotr reference number.

Example:

```
PEA         $000E       ; REF. = 1/4 SEC. IRQ
PUSHLONG    #LABEL      ; HANDLER ADDRESS
_SETVECTOR
```

GetVector Function number = $11

	Input	LongWord	Space for result
	Input	Word	Vector Reference Number
sp—>			
	Output	LongWord	Address
sp—>			

Returns with the vector address for the interrupt manager or handler specified by the vector reference number.

Example:

```
PEA         $0000       ; SPACE FOR RESULT
PEA         $0000
PEA         $0015       ; REF. = 1 SEC. IRQ
_GETVECTOR
```

Vector Reference Numbers:

$0000	Tool Locator #1
$0001	Tool Locator #2
$0002	User's Tool Locator #1
$0003	User's Tool Locator #2
$0004	Interrupt Manager
$0005	COP Manager
$0006	Abort Manager
$0007	System Death Manager
$0008	AppleTalk Interrupt Handler
$0009	Serial Communications Controller Interrupt Handler
$000A	Scan Line Interrupt Handler
$000B	Sound Interrupt Handler
$000C	Vertical Blanking Interrupt Handler
$000D	Mouse Interrupt Handler
$000E	Quarter Second Interrupt Handler
$000F	Keyboard Interrupt Handler
$0010	Front Desk Bus Response Byte Interrupt Handler
$0011	Front Desk Bus SRQ Interrupt Handler
$0012	Desk Accessory Manager
$0013	Flush Buffer Handler
$0014	Keyboard Micro Interrupt Handler
$0015	One Second Interrupt Handler
$0016	External VGC Interrupt Handler
$0017	Other Unspecified Interrupt Handler
$0018	Cursor Update Handler
$0019	Increment Busy Flag (for Scheduler)
$001A	Decrement Busy Flag (for Scheduler)
$001B	Bell Vector (for Sound Tools)
$001C	Break Vector (for Debuggers)
$001D	Trace Vector
$001E	Step Vector
$001F	Reserved Vector
$0020	Reserved Vector
$0021	Reserved Vector
$0022	Reserved Vector
$0023	Reserved Vector
$0024	Reserved Vector
$0025	Reserved Vector
$0026	Reserved Vector
$0027	Reserved Vector
$0028	Control Y Vector
$0029	Reserved Vector
$002A	ProDOS'16 MLI Vector
$002B	OS Vector
$002C	Message Pointer Vector

HeartBeat Tools. These tools allow the application to insert or delete tasks from the HeartBeat queue.

SetHeartBeat Function number = $12

Input LongWord Pointer
sp—>

Installs the task specified by the **pointer** into the HeartBeat queue. The pointer must be set to the address of a task header that precedes the task. The task header area consists of a longword link pointer, count word, and signature word. The link pointer is maintained by the tool, and is set to a value of $00000000 if the task is the last task in the queue. When a task is installed, the link pointer of the previous task is set to point at the task header for the task currently being installed. The count word is set by the application prior to installing the task, and must be maintained by either the task or the application. The count word indicates the number of VBL interrupts that must occur before the associated task is executed. For recurring tasks, the task should reset the count word. For tasks that are run as a software one-shot, the application should reset the count word. The tool will decrement a non zero count word each VBL interrupt. If the decrement results in a count word of zero, the task will be executed. A count word with a value of zero will not be decremented during VBL interrupt, and effectively sets the task inactive until a non zero value is stored to the count word. Tasks are executed in native mode with 8 bit 'm' and 'x'. Task execution should terminate with an 'RTL' instruction. The signature word must be set prior to installing a task, and is used by the tool and the HeartBeat Interrupt Handler to check the integrity of the HeartBeat queue. An example of a HeartBeat task that increments a location in memory everty tenth VBL is shown below:

```
Task1Hdr  Start
          dc  00000000    4i'0'          ; Space for Link Pointer
Task1Cnt  dc  000Y        2i'10'         ; Count word preset to 10
          dc              h,'5AA5'       ; Signature Word $A55A
Task1     anop
          rep             #$20           ; 16 bit 'm'
          longa           on
          phk                            ; data bank = program bank
          plb
          lda             #10            ; reset the task count
          sta             Task1Cnt
          sep             #$20           ; 8 bit 'm'
          longa           off
          lda             >TestLoc       ; and increment an address
          inc             a
          sta             >TestLoc
          rtl
```

The following code will install the task shown above.

```
Install   anop
          PUSHLONG  #LABEL      ; BUFFER ADDRESS
          _SETHEARTBEAT         ; INSTALL TASK
```

Note that when a task is installed into the HeartBeat queue, the HeartBeat Interrupt Handler will automatically be installed into the VBL Interrupt Handler vector. Any handler previously installed in the VBL Interrupt Handler vector will be displaced. Installing a task in the HeartBeat queue does not automatically enable VBL interrupts. It is left to the application to enable VBL interrupts. Also, since tasks are linked with simple pointers, the tasks should reside in 'LOCKED' memory. Tasks that make use of system resources should conform to the protocol set down in the SCHEDULER ERS.

It may be desirable to have a ROM based task executing from a peripheral card. In order to install a ROM based task, twelve bytes of ram must be allocated for use by the task header, with the task executing a jump absolute long to the rom based task. An example of this is shown below:

```
Task1Hdr  dc      4i'0'          ; Space for Link Pointer
Task1Cnt  dc      2i'10'         ; Count word preset to 10
Task1Sig  dc      h,'5AA5'       ; Signature Word $A55A
Task1Jmp  anop
          jmp     >RomTask1      ; jump to ROM based task
```

An example that shows how a program can construct the task header area in RAM for a ROM based task is shown below. Note that this program is run in full native mode (16 bit 'm' and 'x').

```
InstallT1 entry
          lda     #$0001         ; initialize task count
          sta     >Task1Cnt
          lda     #$A55A         ; initialize task signature
          sta     >Task1Sig
          lda     #RomTask1      ; now install 'JMP' to task
          pha
          xba
          and     #$FF00
          ora     #$005C
          sta     >Task1Jmp
          pla
          and     #$FF00
          ora     #^RomTask1
          xba
          sta     >Task1Jmp+2
          PushLong #Label        ; now install the task
          _SetHeartBeat
```

Errors that may occur when installing a task in the HeartBeat queue include:

$0303	Task already installed in queue
$0304	No signature in task header
$0305	Queue has been damaged-task signature missing during search

DelHeartBeat Function number = $13

Input LongWord Pointer
sp—>

Deletes the task specified by the link address from the HeartBeat Interrupt service queue.

Errors that may occur when deleting a task in the HeartBeat queue include:

$0305 Queue has been damaged-task signature missing during search
$0306 Task was not found in queue

Example:

```
PUSHLONG  #LABEL        ; TASK ADDRESS
_DELHEARTBEAT
```

ClrHeartBeat Function number = $14

Clears the HeartBeat queue root link pointer, affectively removing all tasks from the queue.

Example:

```
_CLRHEARTBEAT
```

System Death Manager. This tool call jumps through the system death vector. At system power-up time, a default system death manager is installed into the system death manager vector. The default system death manager will display either a default system death message followed by an error code, or a user defined system death message followed by an error code. The default system death message will display a sliding Apple below a centered default message as shown below:

If a system death call is made with a user defined message, the user defined message will be displayed starting at the upper left hand corner fo the screen. The user defined message may contain up to 254 characters. The text may be moved down by imbedding carriage return characters in the text. Any desired delimiters between the text string and the error code should be included in the text string.

SysDeathMgr Function number = $15

	Input	Word	Error code
	Input	LongWord	Pointer
sp—>			

If the longword pointer is set to zero, the default system death message and the error code passed as the tool input are displayed. If pointer is set to point to an ASCII string, the ASCII string will be displayed with the error code. The first byte of the ASCII string should contain a count equal to the number of characters to be displayed. The ASCII string should have the MSB turned off. Note that this tool call will not return! **Death Messages cannot reside in the Language Card address space.**

Example:

```
PEA         $0004       ; YOUR ERROR CODE
PUSHLONG    #LABEL      ; STRING POINTER
_SYSDEATHMGR
```

Reserved System Death Error Codes:

$0001	ProDOS'16 - Unclaimed interrupt
$0004	Divide by zero
$000A	ProDOS'16 - Volume Control Block unusable
$000B	ProDOS'16 - File Control Block unusable
$000C	ProDOS'16 - Block zero allocated illegally
$000D	ProDOS'16 - Interrupt with I/O shadowing off
$0015	Segment Loader error
$0017-24	Can't load a package
$0025	Out of memory
$0026	Segment Loader error
$0027	File map trashed
$0028	Stack overflow error
$0030	Please insert disk (file manager alert)
$0032-53	Memory manager error
$0100	Can't mount system startup volume

System death error codes above $0100 will be tools specific. The high byte of the error code will contain the tool number reporting the error. The low byte of the error code is defined by the tool set reporting the error. No tool will report an error with the low byte set to a value of $00.

DeathCode	Related ToolSet
$01XX	Tool Locator
$02XX	Memory Manager
$03XX	Miscellaneous Tools
$04XX	Quick Draw
$05XX	Desk Manager
$06XX	Event Manager
$07XX	Scheduler
$08XX	Sound Manager
$09XX	Apple Desktop Bus Tools
$0AXX	SANE
$0BXX	Integer Math Tools
$0CXX	Text Tools
$0DXX	Ram Disk
$0EXX	Menu Manager
$0FXX	Window Manager
$10XX	Control Manager
$11XX	Loader
$12XX	Printer 1
$13XX	Printer 2
$14XX	Line Edit
$15XX	Pick Manager
$16XX	Dialog Manager

GET ADDRESS Tools. These tools are provide to allow an application to determine the address of a parameter used by the system firmware.

GetAddr Function number = $16

	Input	LongWord	Space for result
	Input	Word	Reference number
sp—>			
	Output	LongWord	Pointer to parameter
sp—>			

Parameter reference numbers and parameter size are defined below:

Ref. #	Length	Parameter	
$0000	Byte	IRQ Interrupt Flag	(IRQ.INTFLAG)
$0001	Byte	IRQ Data Flag	(IRQ.DATAREG)
$0002	Byte	IRQ Serial Port 1 Flag	(IRQ.SERIAL1)
$0003	Byte	IRQ Serial Port 2 Flag	(IRQ.SERIAL2)
$0004	Byte	IRQ AppleTalk Flag	(IRQ.APLTLKHI)
$0005	LongWord	Tick Counter	(TICKCNT)
$0006	Byte	IRQ Volume	(IRQ.VOLUME)
$0007	Byte	IRQ Active	(IRQ.ACTIVE)
$0008	Byte	IRQ Sound Data	(IRQ.SOUNDDATA)
$0009	20 Bytes	Variables after a 'BRK'	(BRK.VAR)
$000A	12 Bytes	Event Manager Data	(EVMGRDATA)
$000B	Byte	Mouse Location/Flag	(MouseSlot)
$000C	8 Bytes	Mouse Clamps	(MOUSECLAMPS)
$000D	8 Bytes	Absolute device clamps	(ABSCLAMPS)

Note that parameters with reference numbers from $0000 through $0004 should not be used by applications. These parameters are only valid while servicing an interrupt.

Example:

```
        PEA     $0000       ; SPACE FOR RESULT
        PEA     $0000
        PEA     $000C       ; REF. = MOUSE CLAMPS
        _GETADDR
```

Further definition of some parameters is provided below:

IRQ.INTFLAG	D7	1 = Mouse button down
	D6	1 = Mouse button down on last read
	D5	Status of AN3
	D4	1 = 1/4 second interrupted
	D3	1 = VBL interrupted
	D2	1 = Mega// Mouse switch interrupted
	D1	1 = Mega// Mouse movement interrupted
	D0	1 = System IRQ line is asserted
IRQ.DATAREG	D7	1 = Response byte, 0 = Status byte
	D6	1 = Abort
	D5	1 = Desktop manager sequence pressed
	D4	1 = Flush buffer sequence pressed
	D3	1 = SRQ
	D0-2	0 = No FDB data, 0 ≠ number of valid bytes -1
BRK.VAR	Word	A Register
	Word	X Register
	Word	Y Register
	Word	Stack Pointer
	Word	Direct Register
	Byte	Processor Status
	Byte	Data Bank Register
	Byte	Emulation Flag
	Byte	Program Bank Register
	Word	Program Counter
	Byte	State
	Byte	Shadow
	Byte	CYA
	Byte	MSlot
EVMGRDATA	Word	Journaling flag (JournalFlag)
	LongWord	Pointer to journal driver (JournalPtr)
MouseSlot	Byte	Location of the Mouse (MouseSlot) This is a flag used by the MouseTools. If MouseSlot contains a positive value, then it indicates what slot the mouse resides in. If MouseSlot contains a negative value, the Mouse has not been initialized by the Mouse Tools.

MouseClamps	Word	Low X axis mouse clamp
	Word	Low Y axis mouse clamp
	Word	High X axis mouse clamp
	Word	High Y axis mouse clamp

(Note that setting the mouse clamp values directly is not a viable method of setting the mouse clamps. Setting mouse clamps correctly can only be guaranteed using the mouse tools.)

AbsClamps	Word	Low X axis absolute device clamp
	Word	Low Y axis absolute device clamp
	Word	High X axis absolute device clamp
	Word	High Y axis absolute device clamp

(There is no built in firmware to clamp absolute device position within the absolute device clamp bounds. Absolute device drivers must be responsible for clamping position within the clamp bounds.)

Mouse Tools. These tools are provide to interface with the Mouse. These tools will work with both the built in Front Desk Bus Mouse or the Apple][Mouse. Note that the 'InitMouse' call must be executed first. An error will be returned if a dispatch to the mouse is executed with the mouse firmware switched out.

ReadMouse Function number = $17

	Input	Word	Space for result
	Input	Word	Space for result
	Input	Word	Space for result
sp—>			
	Output	Byte	High Byte X Position
	Output	Byte	Low Byte X Position
	Output	Byte	High Byte Y Position
	Output	Byte	Low Byte Y Position
	Output	Byte	Mouse Status
	Output	Byte	Mouse Mode
sp—>			

Returns Mouse position, status and mode.

Example:

```
PEA         $0000       ; SPACE FOR RESULT
PEA         $0000
PEA         $0000
_READMOUSE
```

InitMouse Function number = $18

	Input	Word	Mouse slot $0000 = Search slots for Mouse $0001-7 = Slot Mouse resides in
sp—>			

Initializes the mouse clamp values to $0000 minimum and $03FF maximum. Mouse mode and status are cleared.

Example:

```
PEA         $0000       ; REQUEST SEARCH
_INITMOUSE
```

SetMouse Function number = $19

	Input	Word	Mode (in low byte)
sp—>			

Mode is set to new value as follows:

$00	Turn off Mouse
$01	Set transparent mode
$03	Set movement interrupt mode
$05	Set button interrupt mode
$07	Set button or movement interrupt mode
$08	Turn mouse off, VBL IRQ active
$09	Set transparent mode, VBL IRQ active
$0B	Set movement interrupt mode, VBL IRQ active
$0D	Set button interrupt mode, VBL IRQ active
$0F	Set button or movement interrupt mode, VBL IRQ active

Example:

```
PEA         $0001       ; TRANSPARENT MODE
_SETMOUSE
```

HomeMouse Function number = $1A

Positions the Mouse at the minimum clamp position.

Example:

```
_HOMEMOUSE
```

ClearMouse Function number = $1B

Sets both the X and Y axis position to $0000 if minimum clamps are negative (delta or relative mode), or to the minimum clamp position if the clamps are positive (absolute mode).

Example:

```
_CLEARMOUSE
```

ClampMouse Function number = $1C

```
        Input       Word        X axis minimum clamp value
        Input       Word        X axis maximum clamp value
        Input       Word        Y axis minimum clamp value
        Input       Word        Y axis maximum clamp value
sp-->
```

Sets the clamp values to new values, and then sets the Mouse position to the miminum clamp values.

Example:

```
        PEA         $0000       ; X MINIMUM
        PEA         $03FF       ; X MAXIMUM
        PEA         $0000       ; Y MINIMUM
        PEA         $03FF       ; Y MAXIMUM
        _CLAMPMOUSE
```

GetMouseClamp Function number = $1D

```
        Input       Word        Space for result
        Input       Word        Space for result
        Input       Word        Space for result
        Input       Word        Space for result
sp-->

        Output      Word        X axis minimum clamp value
        Output      Word        X axis maximum clamp value
        Output      Word        Y axis minimum clamp value
        Output      Word        Y axis maximum clamp value
sp-->
```

Returns the current values of the Mouse clamps.

Example:

```
        PEA         $0000       ; SPACE FOR RESULT
        PEA         $0000
        PEA         $0000
        PEA         $0000
        _GETMOUSECLAMP
```

PosMouse Function number = $1E

```
      Input     Word        X axis position
      Input     Word        Y axis position
sp—>
```

Positions the Mouse to the coordinates specified.

Example:

```
            PEA         $013C       ; X POSITION
            PEA         $028F       ; Y POSITION
            _POSMOUSE
```

ServeMouse Function number = $1F

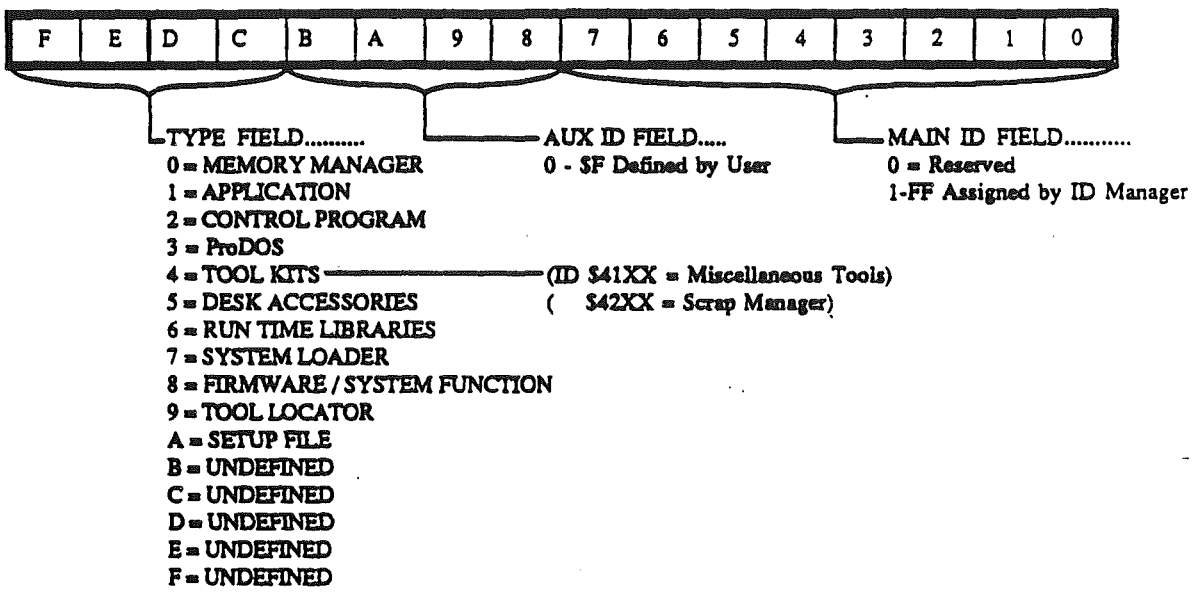
```
      Input     Word        Space for result
sp—>

      Output    Word        Interrupt status (in low byte)
sp—>
```

Returns mouse interrupt status.

Example:

```
            PEA         $0000       ; SPACE FOR RESULT
            _SERVEMOUSE
```

ID Tag Manager. These tools are used to create, delete and inquire status of an ID Tag. The ID Tag is used to mark memory segments as belonging to a specific application or desk accessory. ID tags are made up of three fields encoded in a word parameter. These are the TYPE field, AUX ID field, and MAIN ID field. The type field is encoded in bits 12-14, Aux ID in bits 8-11, and the Main ID in bits 0-7. The AUX ID field is defined by the caller. The Main ID field is generated by the ID Tag manager. The ID Tag will always be assigned with a non zero value in the Main ID field. The Type field has fixed assignments as shown in the table below:



GetNewID Function number = $20

	Input	Word	Space for result
	Input	Word	ID Tag
sp—>			
	Output	Word	ID Tag
sp—>			

Caller passes a full 16 bit ID tag as input with the TYPE defined as the only relevant parameter. The AUX ID field is specified by the caller, and will not be reassigned by the ID manager. The next available MAIN ID will be concatenated to the TYPE and AUX ID fields, and the resulting ID Tag will be returned to the caller. Note that the TYPE field must be non zero. Note that only 255 ID tags can be assigned for any TYPE ID. If an ID cannot be assigned because all the ID tags for that TYPE have been assigned, then an error will be returned indicating that the ID is not available.

Example:

```
PEA     $0000     ; SPACE FOR RESULT
PEA     $5100     ; ITS A DESK ACC.
_GETNEWID
```

DeleteID Function number = $21

	Input	Word	ID Tag
sp—>			

The caller passes the tool a full 16 bit ID tag as input with the TYPE and MAIN ID fields defined as the only relevant parameters. Any ID tags with the same MAIN ID and TYPE are deleted from the current ID tag list. This tool call will not report an error if the tag is not found. It assumes that if its not there, that is what you wanted anyway.

Example:

```
PEA         $5101          ; DELETE DESK ACC TAG
_DELETEID
```

StatusID Function number = $22

	Input	Word	ID Tag
sp—>			

The caller passes the tool a full 16 bit ID tag as input with the TYPE and MAIN ID fields defined as the only relevant parameters. If the ID tag is active, no error will be returned. If the ID tag is inactive, an error will be returned indicating that the ID tag is not available.

Example:

```
PEA         $5101          ; DELETE DESK ACC TAG
_STATUSID
```

Interrupt Control Tools. This tool allows certain interrupt sources to be enabled or disabled.

IntSource Function number = $23

	Input	Word	Source Reference Number
sp—>			

This tool call enables or disables the interrupt source specified by the source reference number. Source reference numbers are shown below:

Ref. #	Source and Action
$0000	Enable Keyboard Interrupts
$0001	Disable Keyboard Interrupts
$0002	Enable Vertical Blanking Interrupts
$0003	Disable Vertical Blanking Interrupts
$0004	Enable Quarter Second Interrupts
$0005	Disable Quarter Second Interrupts
$0006	Enable One Second Interrupts
$0007	Disable One Second Interrupts
$0008	THIS DOES NOTHING
$0009	THIS DOES NOTHING
$000A	Enable FDB Data Interrupts
$000B	Disable FDB Data Interrupts
$000C	Enable Scan Line Interrupts
$000D	Disable Scan Line Interrupts
$000E	Enable External VGC Interrupts
$000F	Disable External VGC Interrupts

Example:

```
PEA         $0002       ; ENABLE VBL IRQ
_INTSOURCE
```

ABOUT KEYBOARD INTERRUPTS.....

When keyboard interrupts are enabled, there is no hardware enable of the keyboard interrupt. The firmware installs a task into the HeartBeat queue and enables VBL interrupts. This causes the HeartBeat interrupt handler to be installed into the VBL interrupt vector. This task will check the status of the keyboard register during each VBL interrupt. If a key is pending, the task will dispatch to the KeyBoard interrupt handler via the keyboard interrupt vector (as installed by the tool 'SETVECTOR'). Since the HeartBeat handler will be installed into the VBL interrupt vector, this precludes the application from installing it's own VBL interrupt handler if keyboard interrupts are to be used. If keyboard interrupts are disabled, the keyboard task is removed from the HeartBeat queue, however the VBL interrupt will not be disabled. If the application wishes to disable keyboard interrupts, and does not wish to have the additional overhead of the VBL interrupts running in the background, the application must disable VBL interrupts also. If no other tasks have been installed into the HeartBeat queue, the additional interrupt overhead is minimal (Interrupt dispatcher and HeartBeat interrupt handler which only increments the tick count before returning).

Firmware Entry Tools. This tool allows the Apple][emulation mode entry points to be supported from full native mode. This tool will preserve the state of the data bank and direct page registers prior to dispatching to the firmware entry point. During the execution of the firmware task, the data bank and direct page registers are set to a value of zero. The data bank and direct page registers are restored on return from the firmware entry point.

FWentry Function number = $24

Input	Word	Space for result	
Input	Word	Space for result	
Input	Word	Space for result	
Input	Word	Space for result	
Input	Word	Accumulator at entry	(low byte only)
Input	Word	X Register at entry	(low byte only)
Input	Word	Y Register at entry	(low byte only)
Input	Word	Emulation mode entry point (16 bits)	

sp—>

Output	Word	Processor status at exit	(low byte only)
Output	Word	Accumulator at exit	(low byte only)
Output	Word	X register at exit	(low byte only)
Output	Word	Y register at exit	(low byte only)

sp—>

This call dispatches to the specified emulation mode entry point with the registers set to the values passed to the tool as input. On return, the register contents resulting from the entry point dispatch will be passed on the stack. Note that only the least significant byte is relevant on the register input and output.

Example:

```
        PEA     $0000       ; SPACE FOR RESULT
        PEA     $0000       ; SPACE FOR RESULT
        PEA     $0000       ; SPACE FOR RESULT
        PEA     $0000       ; SPACE FOR RESULT
        PEA     $0000       ; A REG
        PEA     $0000       ; X REG
        PEA     $0000       ; Y REG
        PEA     $FDDA       ; ENTRY POINT
        _FWENTRY
        BCS     FWERR       ; BRANCH IF ERROR
        PLY                 ; GET FW REGISTERS
        PLX
        PLA
        PLP
        PLP
```

Tick Count Tool. This tool allows caller to read the current value of the tick counter.

GetTick Function number = $25

Input	LongWord	Space for result	
sp—>			
Output	LongWord	Current value of Tick Counter	
sp—>			

Note that the tick count is only incremented by the heartbeat interrupt handler. This means that the heartbeat interrupt handler must be installed, and VBL interrupts must be enabled in order to get an incrementing tick count. Please see the section on heartbeat tasks.

Example:

```
        PEA     $0000       ; SPACE FOR RESULT
        PEA     $0000       ; SPACE FOR RESULT
        _GETTICK
```

PackBytes and UnPackBytes Tools. PackBytes and UnPackBytes provide for the packing and unpacking of any data, but is ususally used for graphic images.

PackBytes Function number = $26

	Input	Word	Space for result
	Input	LongWord	Pointer to pointer to start of area to be packed
	Input	LongWord	Pointer to a word containing size of the area
	Input	LongWord	Pointer to start of the ouput buffer area
	Input	Word	Size of the output buffer area
sp—>			
	Output	Word	Number of packed bytes generated
sp—>			

Upon completion of the call, the pointer to the area to be packed is moved forward to the next packable byte, and the size of area pointed to by the second input parameter is reduced by the number of bytes traversed. An assembly language example follows:

```
*
* PACKBYTES example: Pack a screen image and write it to file "f"
*
PB          START
            lda             #$7D00              ;size of area to pack
            sta             PicSize
            lda             #$E12000            ;addr of screen image
            sta             PicPtr
            lda             #$^E12000
            sta             PicPtr+2
            lda             #buffer             ;pointer to loca buffer
            sta             BufPtr
            lda             #^Buffer
            sta             BufPtr+2

loop        PUSHWORD        #0                  ;Space for result
            PUSHLONG        PicPtr              ;Pointer to data to pack
            PUSHLONG        #PicSize            ;Pointer to word with size of area
            PUSHLONG        BufPtr              ;Pointer to start of output area
            PUSHWORD        BufSize             ;size of output buffer area
            _PACKBYTES
            pla                                 ;get howmuch we did pack this pass
            sta             HowMuch

            CALL            WRITE(f,BufPtr,HowMuch) ;do I/O to write "HowMuch" bytes from "BufPtr" to file "f"

            lda             PicSize             ;see if more to pack;
            bne             loop                ;there is, go back for more
            rts

PicPtr      ds              4                   ;set to $e12000 on entry (screen area)
PicSize     ds              2                   ;size of a picture: set to $7d00 on entry
BufPtr      ds              4                   ;set to point to "Buffer" on entry
BufSize     dc              i2'$400'            ;local buffer for storing packed stuff
HowMuch     ds              2                   ;local storage for value from packbytes
Buffer      ds              $400                ;actual buffer

            END
```

An equivalent example in PASCAL follows:

```
                    .
                    .
                    .
Function packbytes  (  VAR    picptr     : POINTER;
                       VAR    picsize    : POINTER;
                       bufptr    : POINTER;
                       bufsize   : POINTER;
   : INTEGER; EXTERNAL;
                    .
                    .
                    .
picsize := $7D00;
bufsize := $400;    {note: if large enough, could require but one call}
REPEAT
   howmuch := PackBytes (picptr,picsize,bufptr,bufsize);
   write (f,bufptr,howmuch);
UNTIL picsize=0
```

UnPackBytes Function number = $27

	Input	Word	Space for result
	Input	LongWord	Pointer to the buffer containing packed data
	Input	Word	Buffer size
	Input	LongWord	Pointer to pointer to area to unpack data into
	Input	LongWord	Pointer to word containing the size of the area
sp—>			to contain the unpacked data
	Output	Word	Number of bytes unpacked
sp—>			

Upon completion, the pointer to the unpacked data is positioned one past the last upacked byte and the size fo the area is reduced by the amount unpacked. An assembly language example follows:

```
*
* UNPACKBYTES example: UnPack a file "f" onto the screen
*
PB      START

        stz     Mark            ; mark is the file mark we position to
        lda     #$7D00          ;size of area to unpack into
        sta     PicSize
        lda     #$E12000        ;addr of screen image
        sta     PicPtr
        lda     #$^E12000
        sta     PicPtr+2

loop    CALL    SETFILEMARK(f,Mark)     ;position file "f" to position "Mark"

        CALL    READ(f,BufPtr,BufSize)  ;Read BufSize" bytes" into "BufPtr"

        PUSHWORD    #0              ;Space for result
        PUSHLONG    BufPtr          ;Pointer to start of output area
        PUSHWORD    bufsize         ;size of output buffer area
        PUSHLONG    PicPtr          ;Pointer to data to pack
        PUSHLONG    #PicSize        ;Pointer to word with size of area

        _UNPACKBYTES

        pla                         ;get how much we did unpack this pass
        clc                         ;add to previous mark pos.
        sta     Mark
        lda     picsize             ;see if more to unpack;
        beq     done                ;there isn't, so we're done

        CALL    EOF(f)              ;did we get to end of file (safety check)
        bne     loop                ;no, go back for more
Done    rts

BufPtr  dc      i4'Buffer'          ;pointer to buffer area
bufsize dc      i2'$400'            ;local buffer for storing packed stuff
PicPtr  ds      4                   ;set to $e12000 on entry (screen area)
PicSize ds      2                   ;size of a picture: set to $7d00 on entry
Mark    ds      2                   ;file mark position
Buffer  ds      $400                ;actual buffer

        END
```

An equivalent example in PASCAL follows:

```
                    .
                    .
                    .
Function unpackbytes (          bufptr      : POINTER;
                                bufsize     : POINTER;
                        VAR     picptr      : POINTER;
                        VAR     picsize     : POINTER;
   : INTEGER; EXTERNAL;
                    .
                    .
                    .
mark := 0;          {i.e. start of file}
picsize := $7D00
bufsize := $400;    {note: if large enough, could require but one call}
REPEAT
   setfilemark(mark);
   read(f,bufptr,bufsize);
   howmuch := UnPackBytes (bufptr,bufsize,picptr,picsize);
    mark := mark+howmuch;
UNTIL ((picsize=0) or eof(f) );      {eof test in case of bad data}
```

The packed data is in the form of 1 byte containing a flag in the first 2 bits and a count in the remaining 6 bits, followed by one or more data bytes depending on the flags. Their description is as follows:

00XXXXXX : (XXXXXX : 0 -> 63)	= 1 to 64 bytes follow - unique
01XXXXXX : (XXXXXX : 2,4,5 or 6)	= 3,5,6 or 7 repeats of next byte
10XXXXXX : (XXXXXX : 0 -> 63)	= 1 to 64 repeats of next 4 bytes
11XXXXXX : (XXXXXX : 0 -> 63)	= 1 to 64 repeates of next 1 byte taken as 4 bytes (as in '10' case)

Munger. Munger lets you manipulate bytes in a string of bytes. The basic operation is that of searching a destination string for a target string and if found, replacing it with a replacement string. The end of the destination string, if the string is shortened, is padded with a pad character. If the string is elongated, Characters are truncated off of the end. Special cases to allow various other functions are defined below.

Munger		Function number = $28	
	Input	Word	Space for result
	Input	LongWord	Pointer (destptr)
	Input	LongWord	Pointer (destlen)
	Input	LongWord	Pointer (targptr)
	Input	Word	Integer (targlen)
	Input	LongWord	Pointer (replptr)
	Input	Word	Integer (repllen)
	Input	LongWord	Pointer (pad)
sp—>			
	Output	Word	Amount of Pad / Truncations
sp—>			

Where input is:

destptr:	Pointer to pointer to the text to be manipulated
destlen:	Pointer to number of bytes to manipulate
targptr:	Pointer to string to be searched for from destptr
targlen:	Number of bytes for targptr
replptr:	Pointer to string to replace when targptr found
repllen:	Number of bytes for replptr
pad:	Character value to pad shortened input with

And output is:

destptr:	Updated to one past end of any replacement
destlen:	Old value reduced by bytes scanned across
pad:	Number of bytes padded (or truncated)
Munger:	Zero if target found, negative if not

Special cases:

If targptr is 0, the substring of length targlen is replaced by the replptr string.

If targlen is 0, replptrs string is inserted at destptr.

If replptr is 0, destptr is updated to past the end of the match of the targptr string.

If repllen is 0, (and replptr is not) the targptr string is deleted rather than replaced (since the replacement string is empty).

There is one case in which munger performs a replacement even if it doesn't find all for the target string. If the destptr string in ints entirety is at the beginning of the targptr string, then the destptr string is totally replaced by the replptr string.

```
*
* MUNGER example : editing a line of text
*
* Changes "robert irwin eagle toramaga marcia houdini berns"
*   into "robert irwin EAGLE toramaga marcia houdini berns"
*
MG      START

        lda     #Name           ;set pointer to name
        sta     DestPtr
        lda     #^Name
        sta     DestPtr+2

        lda     #48             ;get length
        sta     DestLen

        PUSHWORD #0             ;space for result
        PUSHLONG DestPtr        ;pointer to textstring to manipulate
        PUSHLONG #DestLen       ;Pointer to word with number bytes to change
        PUSHLONG #eagleLC       ;Points to "eagle" (lower case)
        PUSHWORD #5             ;"eagle" has 5 letters
        PUSHLONG #eagleUC       ;Pointer to "EAGLE" (upper case)
        PUSHWORD #5             ;"EAGLE" has 5 letters
        PUSHLONG #PAD           ;Pad char (don't care for this example)

        _MUNGER

        pla                     ;[THIS WILL BE ZERO, AS WILL PAD]

        rts

DestPtr ds      2               ;on entry, will point to name
DestLen ds      2               ;on entry will be set to "NLen"

PAD     ds      2               ;pad value

eagleLC dc      c'eagle'
eagleUC dc      c'EAGLE'

name    dc      c'robert irwin eagle toramaga marcia houdini berns'
```

An equivalent example in Pascal follows:

```
                    .
                    .
                    .
Function munger     ( VAR    destptr     : POINTER;
                      VAR    destlen     : INTEGER;
                             targptr     : POINTER;
                             targlen     : INTEGER;
                             replptr     : POINTER;
                             repllen     : INTEGER;
                      VAR    PAD         : INTEGER;
  : INTEGER; EXTERNAL;
                    .
                    .
                    .
```

{segment to replace a word in lower case with it's upper case equivalent}

```
name := 'robert irwin eagle toranoga marcia houdini berns';
I:= LEN (name);
i := munger(name, I,'eagle',5,'EAGLE',5,p);
```

{upon completion, *i* is 0, *p* is 0, and *name* is 'robert irwin EAGLE toranaga marcia houdini berns'}

Interrupt Enable State Tool. This function returns with the state of hardware interrupt enable states for interrupt sources that can be controlled by the miscellaneous tool set.

GetIRQenbl Function number = $29

sp—>	Input	Word	Space for result
sp—>	Output	Word	Status of hardware interrupt enables

Status in returned word is defined below:

D8-15	Undefined
D7	1 = Keyboard interrupts enabled
D6	1 = Vertical blanking interrupts enabled
D5	1 = Quarter second interrupts enabled
D4	1 = One second interrupts enabled
D3	Reserved
D2	1 = Front Desk Bus data interrupts enabled
D1	1 = Scan line interrupts enabled
D0	1 = External VGC interrupts enabled

Example:

```
        PEA         $0000           ; SPACE FOR RESULT
        _GETIRQENBL
```

SetAbsClamp Function number = $2A

```
        Input       Word        X axis minimum clamp value
        Input       Word        X axis maximum clamp value
        Input       Word        Y axis minimum clamp value
        Input       Word        Y axis maximum clamp value
sp—>
```

Sets the clamp values for absolute devices to new values.

Example:

```
            PEA         $0000       ; X MINIMUM CLAMP
            PEA         $03FF       ; X MAXIMUM CLAMP
            PEA         $0000       ; Y MINIMUM CLAMP
            PEA         $03FF       ; Y MAXIMUM CLAMP
            _SETABSCLAMP
```

GetAbsClamp Function number = $2B

```
        Input       Word        Space for result
        Input       Word        Space for result
        Input       Word        Space for result
        Input       Word        Space for result
sp—>

        Output      Word        X axis minimum clamp value
        Output      Word        X axis maximum clamp value
        Output      Word        Y axis minimum clamp value
        Output      Word        Y axis maximum clamp value
sp—>
```

Returns the current values of the absolute device clamps.

Example:

```
            PEA         $0000       ; SPACE FOR RESULT
            PEA         $0000
            PEA         $0000
            PEA         $0000
            _GETABSCLAMP
```

Miscellaneous Tool Set Error Codes

$0000	No Error
$0301	Bad Input Parameter
$0302	No Device for Input Parameter
$0303	Task is already in Heartbeat queue
$0304	No signature in task header was detected during insert or delete
$0305	Damaged queue was detected during insert or delete
$0306	Task was not found during delete
$0307	Firmware task was unsuccessful
$0308	Detected damaged HeartBeat Queue
$0309	Attempted dispatch to a device that is not connected
$030A	Undefined
$030B	ID tag not available

Summary of functions within the Miscellaneous Tool Set

Function	Number	Description
$01	1	MTBootInit
$02	2	MTStartUp
$03	3	MTShutDown
$04	4	MTVersion
$05	5	MTReset
$06	6	MTStatus
$07	7	MTSpare1
$08	8	MTSpare2
$09	9	WriteBRam
$0A	10	ReadBRam
$0B	11	WriteBParam
$0C	12	ReadBParam
$0D	13	ReadTimeHex
$0E	14	WriteTimeHex
$0F	15	ReadAsciiTime
$10	16	SetVector
$11	17	GetVector
$12	18	SetHeartBeat
$13	19	DelHeartBeat
$14	20	ClrHeartBeat
$15	21	SysDeathMgr
$16	22	GetAddr
$17	23	ReadMouse
$18	24	InitMouse
$19	25	SetMouse
$1A	26	HomeMouse
$1B	27	ClearMouse
$1C	28	ClampMouse
$1D	29	GetMouseClamp
$1E	30	PosMouse
$1F	31	ServeMouse
$20	32	GetNewID
$21	33	DeleteID
$22	34	StatusID
$23	35	IntSource
$24	36	FWentry
$25	37	GetTick
$26	38	PackBytes
$27	39	UnPackBytes
$28	40	Munger
$29	41	GetIRQenbl
$2A	42	SetAbsClamp
$2B	43	GetAbsClamp

ERROR CODES