SmartPort Extensions

Revision History.....

January 31, 1986	Ver. 00.10	R. Montagne	Device Information Block status call was updated to show bit 7 of the device subtype being used to indicate whether a device is an Extended device (page 11).
March 17, 1986	Ver. 00.20	R. Montagne	All references to the Protocol Converter have been changed to reference SmartPort.

Based on a novel by Mike Askins
Written by Ray Montagne

PREFACE

In order to support future products within the Apple// product line, it has become apparent that the SmartPort, as it exists now does not provide the flexibility necessary to grow with the product line. The SmartPort extensions described within this document are intended to provide a means of supporting Cortland, the Apple// SCSI interface card, and other future products. The SmartPort has been extended in three areas. The Block Address Field passed in the parameter list when a block device access is requested, has been expanded to support a four byte block address. In addition, the Buffer Address Field passed in the parameter list has been expanded to support a four byte buffer address. The Parameter List Pointer which follows the command byte just after the 'JSR' to the SmartPort driver has also been expanded to a four byte address field. The command packet format has also been changed to provide for these extensions. The command packet extensions have been made in such a manner that current non extended devices will not be affected, and should not respond to packets using the extended protocol. New extended devices will be able to arbitrate between non extended and extended formats.

Each of these extensions is described in detail within this document. This document should be considered as an addendum to the current SmartPort specification. In order to maintain compatability with the current Apple// product line, any devices designed to this specification should also adhere to the current non extended SmartPort and CBUS architecture, although it is not necessary for a non extended device to support the extended format.

GENERAL

The SmartPort provides a method of attaching a series of devices to the external disk port of the Apple//c and Cortland, or an Apple//e with SmartPort interface card. When devices are connected to the CPU with the disk port configured as a bus, the disk port is refered to as CBUS. Future CPU's within the Apple// product line may support the SmartPort with either an external interface card, or through implementing CBUS with the built in disk support. The SmartPort is a program that converts the calls màde to it into a format which can be transmitted over CBUS. An exception would be an Apple// peripheral card that supports a specific type of device via the SmartPort interface while not actually using the CBUS. The SCSI Interface card for the Apple// is an example of this. Within the SmartPort, calls fall into one of two groups. Extended calls, and non extended calls. For detailed information on non extended calls, refer to the SmartPort specification.

To use the extended SmartPort interface, a program issues calls in a manner similar to that used for ProDOS Machine Language Interface (MLI) calls. The topmost 'level' of one of these calls is a 'JSR' to the SmartPort entry point (DISPATCH), followed by a single byte which specifies the SmartPort extended command type, followed by a long word address (low word, high word) of a table which contains the parameters necessary for the call. Here is an example of a call:

```
SP_CALL   JSR   DISPATCH      ;Call SmartPort command dispatcher
          DFB   CMDNUM        ;This specifies the command type
          DW    CMDLIST       ;Low word pointer to the parameter list
          DW    ^CMDLIST      ;High word pointer to the parameter list
          BCS   ERROR         ;Carry is set on an error
```

Upon completion of the call, execution returns to the RTS address plus five (the 'BCS' statement in this example). If the call was successful, the C flag is cleared, and the A register is set to 0. If the call was unsuccessful, the C flag is set and the A register contains the error code. The complete register status upon completion is summarized below.

REGISTER STATUS ON RETURN FROM SMARTPORT

	65816 Status byte NVmxDIZC	Acc	Xreg	Yreg	PC	SP
Successful Call	XX110UX0	0	n	n	JSR+5	U
Unsuccessful Call	XX110UX1	Error	X	X	JSR+5	U

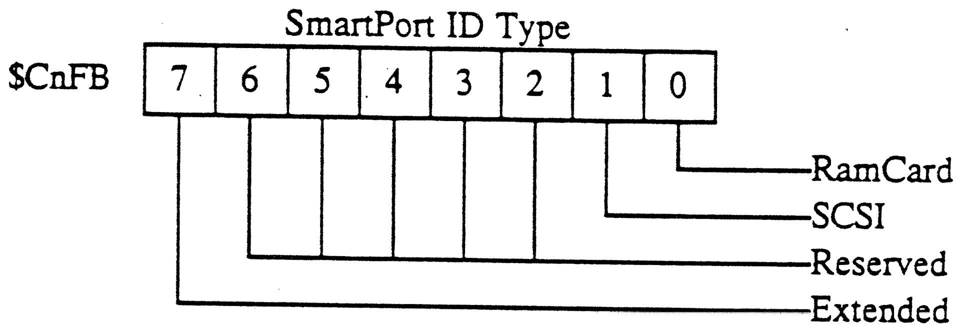
(Note: X = undefined, U = unchanged, n = undefined for tranfers to the device or number of bytes transferred when the transfer was from the device to the host)

Determining the DISPATCH address

Regardless of whether you are operating in a machine with built in disk support or a machine containing cards supporting the SmartPort Interface, the existance of the SmartPort Interface may be determined by examining the signature bytes as follows;

$Cn01 = $20 $Cn03 = $00 $Cn05 = $03 $Cn07 = $00

where n = the slot number. All cards or built in slot support firmware with these values support SmartPort calls. In addition, the Protocol Convert ID Type byte can be examined to obtain more information about what special support may be built into the SmartPort firmware driver. The Protocol Convert ID Type byte located at $CnFB has been encoded to indicate the type of devices that can be supported by the SmartPort firmware. Note that a driver that supports the extended modes will also support the existing non extended modes.



Once having determined that a SmartPort interface exists in a slot or port, you need to determine the entry point, or DISPATCH address, for this slot or port. This address is determined by the <u>value</u> found at $CnFF, where n is the slot number. By adding the value at $CnFF to $Cn00, one obtains the standard <u>ProDOS</u> entry point for block devices. Information about this entry point is described in the ProDOS Technical Reference manual. The SmartPort entry point, DISPATCH, is located three bytes after the ProDOS entry point. Therefore, the SmartPort entry point is $Cn00 plus 3 plus the <u>value</u> found at $CnFF.

For example, if a the signature bytes for the SmartPort interface are found in slot 5, and $C5FF contains a $0A, the ProDOS entry point will be $C50A, and the SmartPort entry point is three larger than $C50A, or $C50D.

Dispatches to the SmartPort with a CPU based on the 65816 should be made in emulation mode.

ProDOS Interface

A new implementation of ProDOS called ProDOS'16 will be available for the Cortland CPU. This new implementation will recognize and be able to communicate directly with a SmartPort driver. The current implementation of ProDOS does not recognize SmartPort Interfaces. In order to have SmartPort interfaces supported by ProDOS, it is necessary to support the ProDOS entry point as well as ProDOS commands.

ProDOS Limitations

With ProDOS, it is not possible for a peripheral card to be recognized as both a character device driver, and a block device driver. Thus it is necessary to implement the ProDOS driver as the type of device most likely to be used with the SmartPort Interface. If block devices are most prevelant on the SmartPort Interface, then the ProDOS driver should be implemented as a block device. If character devices are most prevelant on the SmartPort Interface, then the ProDOS driver should be implemented as a character device. Of course the draw back of this is that only one type of device may be supported by ProDOS. This will not be a problem with ProDOS'16 since it will communicate directly with the SmartPort Interface.

ProDOS Implementations

ProDOS entry points are best supported by converting the input to the ProDOS driver into a format compatable with the non extended SmartPort, and then have the ProDOS driver call the SmartPort driver. The non extended format is chosen because the majority of devices support the non extended protocol rather than the extended format. Also, ProDOS is runs in a 6502 environment that may not easily support direct tranfers between the SmartPort and any bank of memory. On return from the SmartPort, the ProDOS driver must map all non fatal SmartPort errors into a non error condition (ACC=$00 with carry clear). In addition, any fatal SmartPort error must be converted by the ProDOS driver into standard ProDOS error codes.

Boot Devices

In order to support the boot entry point at $Cn00, the device driver firmware must store $Cn in location $07F8 (MSLOT), and then read block $00000000 into memory at $00000800. After the block read, the following conditions must be met before the boot process can continue:

$00000800 = $01
$00000801 ≠ $00

If the above prescribed conditions are met, the boot driver should load the X register with a value equal to the slot number times sixteen and then execute a jump to location $00000801 to complete the boot process. If the boot conditions are not met, then the error condition must be handled in one of two ways depending on how the boot was initiated. The driver can determine how the boot was initiated by examining the word value at location $00 in absolute zero page.

If $00000000 = $Cn00, then the boot was initiated by an AutoScan
If $00000000 ≠ $Cn00, then the boot was initiated by PR#n or Cn00G

Exiting the boot driver from a boot initiated by an AutoScan simply requires that the boot driver execute a jump to location $0000FABA. Exiting the boot drive from a boot that was not initiated by an AutoScan requires several tasks be executed. The I/O hooks should be reset to the system defaults, text mode should be set followed by a screen home. Then an error should be displayed that describes why the boot failed prior to executing a jump back to basic. An example is shown below;

```
LOC0        EQU     $00             ; AUTOSCAN BOOT ENTRY VECTOR
LOC1        EQU     $01
MSLOT       EQU     $07F8           ; $Cn STORED HERE BE DRIVER AT ENTRY
SETVID      EQU     $FE93           ; SET VIDEO HOOKS BACK
SETKBD      EQU     $FE89           ; SET KEYBOARD HOOKS BACK
TEXT        EQU     $FB39           ; SET TEXT MODE
HOME        EQU     $FC58           ; CLEAR SCREEN & HOME CURSOR
BASIC       EQU     $E000           ; BASIC ENTRY POINT
AUTOSCAN    EQU     $FABA           ; AUTOSCAN LOOP
STACK       EQU     $0100           ; STACK PAGE

BOOT        PHP                     ; DISABLE INTERRUPTS WHILE GETTING $Cn
            SEI
            JSR     KNOWNRTS        ; GET $Cn (this JSR must be in $CnXX)
            TSX                     ; GET STACK POINTER
            LDA     STACK,X         ; GET $Cn OFF STACK PAGE
            STA     MSLOT           ; AND SAVE IT AWAY
            PLP                     ; RESTORE INTERRUPT STATE
            JSR     READBLK0        ; READ BLOCK 0 INTO $800
            LDX     $0800           ; CHECK BOOT BLOCK
            DEC
            BNE     BOOTFAIL        ; IF $800 ≠ $01, THEN BOOT ERROR
            LDA     $0801
            BEQ      BOOTFAIL          ; IF $801 = $00, THEN BOOT ERROR
            LDA      MSLOT             ; GET $Cn
            ASL      A                 ; MAKE $n0
            ASL      A
            ASL      A
            ASL      A
            TAX                        ; X MUST HAVE $n0
            JMP      $0801             ; IF IT'S OK, JUMP TO BOOT CODE

BOOTFAIL    LDA      LOC0              ; WHERE DID BOOT COME FROM
            BNE      NOAUTO            ; IF $00 ≠ $CN00 THEN NOT AUTOSCAN
            LDA      LOC1
            CMP      MSLOT
            BNE      NOAUTO            ; IF $01 ≠ MSLOT THEN NOT AUTOSCAN
            JMP      AUTOSCAN          ; CONTINUE SLOT SCAN

NOAUTO      JSR      SETVID            ; SET VIDEO HOOKS BACK
            JSR      SETKBD            ; SET KEYBOARD HOOKS BACK
            JSR      TEXT              ; SET TEXT MODE
            JSR      HOME              ; CLEAR SCREEN & HOME CURSOR
            JSR      ERROR             ; DISPLAY ERROR MESSAGE
            JMP      BASIC             ; AND OFF TO BASIC

KNOWNRTS    RTS
```

CMDNUM and CMDLIST

The CMDNUM specifies the command type and is in the range of $00 - $09 for non extended commands, or $40 - $49 for extended commands. The parameter list, CMDLIST, varies for each call, though the first byte in the list always specifies the number of parameters in the call. This is NOT the physical number of bytes; it is the logical number of pieces of information passed in the list. A summary of command numbers and their associated parameter lists can be found in an appendix.

Unit Numbers

A part of every parameter list is the Unit number. The unit number specifies which device connected to the SmartPort will respond to the command you are giving. This unit number is distinct from the normal ProDOS unit number. The SmartPort maps devices sequentially; Unit $01 is the first device in the chain, $02 is the second, etc. Calls which allow you to reference the SmartPort itself require the specification of Unit $00 (Status, Init, and Control allow you to do this).

Determining the Number of Devices

You can determine the number of devices connected to the SmartPort by executing a SmartPort status call (described later) with a unit number of $00. Some cards will have only one unit you can reference. Others may have more than one unit.

SmartPort Limitations

There are two major constraints on the use of the SmartPort. The first is that its stack usage varies with the implementation of the SmartPort Interface. The non extended implementations on both the Apple//c and Apple// SmartPort Interface card use between 30 and 35 bytes of stack. It is not clear at this time how much stack is required by other implementations of the SmartPort, however any future implementations should attempt to not use any more stack space than 30-35 bytes. Programs should allow this much stack space on a call.

Finally, the SmartPort cannot generally be used to put anything into absolute Zero Page locations. With new CPU's using the 65816, it is possible to locate zero page anywhere within address bank $00 through use of the direct register. Absolute zero page is defined as zero page when the direct register has been set to $0000.

EXTENDED STATUS CMDNUM = $40

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit number
	Byte 2 :	status list pointer (low byte, low word)
	Byte 3 :	status list pointer (high byte, low word)
	Byte 4 :	status list pointer (low byte, high word)
	Byte 5 :	status list pointer (high byte, high word)
	Byte 6 :	status code

This call returns the status information about a particular device or the SmartPort itself. There are defined status calls for returning general information. In general, device specific calls can be implemented by a device for diagnostic or other information.

On return from a status call, the X and Y registers contain a count of the number of bytes tranferred to the host. X contains the low byte of the count, while Y contains the high byte value of the count.

Required Parameters

unit num: 1 byte value
Range : $00, $01 - $7E

This is the unit number. Each device has a unique number assigned to it at initialization time. The numbers are assigned according to the device's position in the chain. A status call with a unit number of $00 specifies a call for the overall SmartPort status. (The status list returned on this call is explained below.)

status list: pointer

This is a long word pointer to the buffer where the status is to be returned. Note that the length of the buffer will vary depending on the status request being made.

status code: 1 byte value
Range: $00 - $FF

This is the number of the status request being made. All devices respond to the following requests:

Code	Status returned
$00	Return device status
$01	Return device control block
$02	Return newline status (character devices only)
$03	Return device information block (DIB)

Although devices must respond to the status requests listed above, the device may not be able to support the request. In this case, the device should return an Invalid Status Code error ($21).

Statcode = $00
The device status consists of four bytes. The first is the general status byte:

Bit	Function
7	1 = Block device, 0 = Character device
6	1 = Write allowed
5	1 = Read allowed
4	1 = Device online, or disk in drive
3	1 = Format allowed
2	1 = Media Write Protected (block devices only)
1	1 = Device currently interrupting (not supported on Cortland)
0	1 = Device currently open (character devices only)

If the device is a block device, the next four bytes are the size in 512 byte blocks. The least significant byte is first. The maximum size is $FFFFFFFF blocks, or about 2.2 trillion bytes. If the device is a non block device, these bytes are set to zero.

Statcode = $01
The device control block is device dependent. The DCB is typically used to control various operating characteristics in a device. The DCB is set with the corresponding control call. The first byte will be the number of bytes in the control block. A value of $00 returned in this byte should be interpreted as a DCB length of 256, while a value of $01 would be a DCB length of 1 byte. The length of the DCB will always be in the range of 1 to 256 bytes excluding the count byte.

Statcode = $02
There are currently no character devices implemented for use on the SmartPort, and therefore the Newline status is presently undefined. Since the Apple// SCSI Interface card can support both character devices and block devices, and utilizes the SmartPort interface, some definition may come out of that design.

Statcode = $03
This call returns the device's information block. It contains information identifying the device, its type, and various other attributes. The returned status list has the following form:

```
STATLIST  DFB  DEVICE_STATBYTE1   ;Same as byte 1 in status code = $00
          DW   DEVICE_SIZE_LO     ;Number of blocks on device
          DW   DEVICE_SIZE_HI     ;Number of blocks on device
          DFB  ID_STR_LEN         ;Length in bytes (16 max)
          ASC  'device name'      ;(16 bytes) upper case ascii, msb=0, blanks added
          DFB  DEVICE_TYPE        ;
          DFB  DEVICE_SUBTYPE     ;Bit 7 indicates Extended Device
          DW   VERSION            ;Device firmware version number
```

Note that the first five bytes of the DIB are the same bytes returned in the Device Status call. Note also that since the name string is always 16 characters, the position of the bytes in the list is always fixed relative to the beginning of the DIB. Bit 7 of the device subtype byte returned in the DIB is used to indicate whether a device is an extended device. If bit 7 = 0, then the device is non extended. If bit 7 = 1, then the device is extended.

<u>SmartPort Status</u>

A status call with a unit number of $00 and a status code of $00 is a request to return the status of the SmartPort <u>as a whole</u>. The number of devices as well as the current interrupt status is returned. The Format of the status list returned is as follows:

STATLIST	byte 0:	Number of devices
	byte 1:	Interrupt Status (Bit 7 clear => no interrupt)
	byte 2:	Reserved
	byte 3:	Reserved
	byte 4:	Reserved
	byte 5:	Reserved
	byte 6:	Reserved
	byte 7:	Reserved

The number of devices byte tells the caller the total number of devices connected to this slot or port. This number will always be in the range of 0 to 127.

The interrupt status byte is used by programs to determine if the SmartPort was the source of an interrupt. If the most significant bit of this byte is set, there is a device (or devices) in the chain that requires interrupt servicing. Which device is actually interrupting cannot be determined from this value. The user's interrupt handler, having determined that a SmartPort interrupt has occured must poll each device on the chain to find out which device requires service.

The Extended SmartPort interface is currently not supported on either the Apple//c or the Apple// SmartPort Interface card, however, if future revisions of these products were to implement the Extended SmartPort, interrupts should be handled as they currently are with the non extended implementation.

Possible Errors

$06	BUSERR	Communications error
$21	BADCTL	Invalid status code
$30-$3F	$50-$7F	Device specific error

(Some user defined status calls may use other error codes.)

EXTENDED READ BLOCK cmdnum = $41

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	data_buffer pointer (low byte, low word)
	Byte 3 :	data_buffer pointer (high byte, low word)
	Byte 4 :	data_buffer pointer (low byte, high word)
	Byte 5 :	data_buffer pointer (high byte, high word)
	Byte 6 :	block_num (low byte, low word)
	Byte 7 :	block_num (high byte, low word)
	Byte 8 :	block_num (low byte, high word)
	Byte 9 :	block_num (high byte, high word)

This call reads one 512 byte block from the block device specified by unit_num into memory starting at the address specified by data_buffer.

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: pointer

This is a longword pointer to the user's buffer that the data is to be read into. The buffer must be 512 or more bytes in length.

block num: LongWord number

This is the logical address of a block of data to be read. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical block is performed by the device. (Note that this is a four byte number.)

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2D	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off line or no disk in drive

EXTENDED WRITE BLOCK cmdnum = $42

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	data_buffer pointer (low byte, low word)
	Byte 3 :	data_buffer pointer (high byte, low word)
	Byte 4 :	data_buffer pointer (low byte, high word)
	Byte 5 :	data_buffer pointer (high byte, high word)
	Byte 6 :	block_num (low byte, low word)
	Byte 7 :	block_num (high byte, low word)
	Byte 8 :	block_num (low byte, high word)
	Byte 9 :	block_num (high byte, high word)

This call writes one 512 byte block to the block device specified by unit_num from memory starting at the address specified by data_buffer.

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: pointer

This is a longword pointer to the user's buffer that the data is to be read into. The buffer must be 512 or more bytes in length.

block num: LongWord number

This is the logical address of a block of data to be written. There is no general connection between block numbers and the layout of tracks and sectors on the disk. The translation from logical to physical block is performed by the device. (Note that this is a four byte number.)

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2B	NOWRITE	Disk write protected
$2D	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off line or no disk in drive

EXTENDED FORMAT cmdnum = $43

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call formats a block device. It should be noted that the format done by this call is NOT linked to any operating system: it simply prepares all blocks on the medium for reading and writing. Operating system specific catalog information such as bitmaps and catalogs are not laid down by this call.

Required parameters

unit num:	1 byte value
	Range: $01-$7E

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2B	NOWRITE	Disk Write Protected
$2F	OFFLINE	Device off line or no disk in drive

EXTENDED CONTROL cmdnum = $44

CMDLIST	Byte 0 :	parameter count = 3
	Byte 1 :	unit_num
	Byte 2 :	control_list (low byte, low word)
	Byte 3 :	control_list (high byte, low word)
	Byte 4 :	control_list (low byte, high word)
	Byte 5 :	control_list (high byte, high word)
	Byte 6 :	control_code

This call sends control information to the device. The information can be either general or device specific. A control call with a unit number of zero has special significance as noted below.

Required parameters

unit num: 1 byte value
Range: $01-$7E

control list: pointer

This is a longword pointer to the user's buffer where the control information is to be read from. The first two bytes specify the length of the control list; the low byte is first. A control list is mandatory even if the call being issued does not pass information in the list. A length of zero is used for the first two bytes in this case.

control code: 1 byte value
Range: $00-$FF

This is the number of the control request being made. This number and function is device specific, with the exception that all devices must reserve the following codes for specific functions.

Code	Control Function
$00	Reset the device.
$01	Set device control block
$02	Set newline status (character devices only)
$03	Service device interrupt

Code = $00
Performs a soft reset of the device. Generally returns 'housekeeping' values to some reset value.

Code = $01
Allows the user to set the device control block. Devices generally use the bytes in this block to control global aspects of the device's operating environment. Since the length is device dependent, the recommended way to set the DCB is to first read in the DCB (with the STATUS call), alter the bits of interest, and then write out the same string with this call. The first byte is the length of the DCB (excluding the byte itself). A value of $00 in the length byte corresponds with a DCB size of 256 bytes, while a count value of $01 corresponds with a DCB size of 1 byte. A count value of $FF corresponds with a DCB size of 255 bytes.

Unit $00 Control Calls

A control call with a unit number of $00 specifies that the call applies to the SmartPort as a whole. There are two calls currently available.

Code = $00 Enable Interrupts from the SmartPort
This call is used to enable interrupt hardware related to the SmartPort Interface. This call is not supported by the built in SmartPort support on Cortland, and will return a bad control code error ($21).

Code = $01 Disable Interrupts from the SmartPort
This call is used to disable interrupt hardware related to the SmartPort Interface. This call is not supported by the built in SmartPort support on Cortland, and will return a bad control code error ($21).

Possible Errors

$06	BUSERR	Communications error
$21	BADCTL	Invalid control code
$22	BADCTLPARM	Invalid parameter list
$30-$3F		Device specific error

EXTENDED INIT cmdnum = $45

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call provides the application with a way of resetting the SmartPort.

Required parameters

unit num:	1 byte value
	Value: $00

The SmartPort will go through it's initialization sequence, hard resetting all devices and sending each their device numbers. This call is made internally at first access and it should never be necessary for an application to make this call. (Though it is never recommended to connect new devices with the CPU power on, this call provides a method for the SmartPort to communicate with devices connected midstream.)

Possible Errors

$06	BUSERR	Communications error
$28	NODRIVE	No Device Connected

EXTENDED OPEN cmdnum = $46

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call is used to prepare a character device for reading or writing.

Note that extended block devices do not accept this call, and will return a invalid command error ($01).

Required parameters

unit_num: 1 byte value
Range: $01-$7E

Possible Errors

$01	BADCMD	Invalid command
$06	BUSERR	Communications error
$28	NODRIVE	No Device Connected

EXTENDED CLOSE cmdnum = $47

CMDLIST	Byte 0 :	parameter count = 1
	Byte 1 :	unit_num

This call is used to tell an extended character device that a sequence of reads or writes is over. In the case of a printer, this call could have the effect of flushing the print buffer.

Note that extended block devices do not accept this call, and will return a invalid command error ($01).

Required parameters

unit num: 1 byte value
Range: $01-$7E

Possible Errors

$01	BADCMD	Invalid command
$06	BUSERR	Communications error
$28	NODRIVE	No Device Connected

EXTENDED READ cmdnum = $48

CMDLIST	Byte 0 :	parameter count = 4
	Byte 1 :	unit_num
	Byte 2 :	data _ buffer pointer (low byte, low word)
	Byte 3 :	data _ buffer pointer (high byte, low word)
	Byte 4 :	data _ buffer pointer (low byte, high word)
	Byte 5 :	data _ buffer pointer (high byte, high word)
	Byte 6 :	byte_count low
	Byte 7 :	byte_count high
	Byte 8 :	address pointer (low byte, low word)
	Byte 9 :	address pointer (high byte, low word)
	Byte 10 :	address pointer (low byte, high word)
	Byte 11 :	address pointer (high byte, high word)

This call reads a number of bytes from the device specified by unit_num into memory starting at the address specified by data_buffer. The meaning of the address parameter depends on the device involved. Although this call is generally intended for use by extended character devices, an extended block device might use this call to read a block of a non standard size (greater than 512 bytes per block).

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: LongWord pointer

This is the four byte pointer to the user's buffer that the data is to be read into. The buffer must be large enough to contain the number of bytes requested.

byte count: 2 byte number

This specifies the number of bytes which are to be transferred. All of the current implementations of the SmartPort utilizing CBUS have a limitation of 767 bytes for this call. Other peripheral cards supporting the SmartPort interface, and using this call may not have this limitation.

address: LongWord

This is a device specific parameter. An example of how this call might be implemented with an extended block device, is to use the address as a block address for accessing a non standard block (that is, to access a block larger than 512 bytes).

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2B	NOWRITE	DISK WRITE PROTECTED
$2F	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off line or no disk in drive

EXTENDED WRITE cmdnum = $49

CMDLIST	Byte 0 :	parameter count = 4
	Byte 1 :	unit_num
	Byte 2 :	data _ buffer pointer (low byte, low word)
	Byte 3 :	data _ buffer pointer (high byte, low word)
	Byte 4 :	data _ buffer pointer (low byte, high word)
	Byte 5 :	data _ buffer pointer (high byte, high word)
	Byte 6 :	byte_count low
	Byte 7 :	byte_count high
	Byte 8 :	address pointer (low byte, low word)
	Byte 9 :	address pointer (high byte, low word)
	Byte 10 :	address pointer (low byte, high word)
	Byte 11 :	address pointer (high byte, high word)

This call writes a number of bytes to the device specified by unit_num from memory starting at the address specified by data_buffer. The meaning of the address parameter depends on the device involved. Although this call is generally intended for use by extended character devices, an extended block device might use this call to write a block of a non standard size (greater than 512 bytes per block).

Required parameters

unit num: 1 byte value
Range: $01-$7E

data buffer: LongWord pointer

This is the four byte pointer to the user's buffer that the data is to be written from. The buffer must be large enough to contain the number of bytes requested.

byte count: 2 byte number

This specifies the number of bytes which are to be transferred. All of the current implementations of the SmartPort utilizing CBUS have a limitation of 767 bytes for this call. Other peripheral cards supporting the SmartPort interface, and using this call may not have this limitation.

address: LongWord

This is a device specific parameter. An example of how this call might be implemented with an extended block device, is to use the address as a block address for accessing a non standard block (that is, to access a block larger than 512 bytes).

Possible Errors

$06	BUSERR	Communications error
$27	IOERROR	I/O Error
$28	NODRIVE	No Device Connected
$2B	NOWRITE	DISK WRITE PROTECTED
$2F	BADBLOCK	Invalid block number
$2F	OFFLINE	Device off line or no disk in drive

ProDOS Entry Point

The ProDOS entry point can be thought of as simply a 'front end' to the non extended SmartPort which is tuned to the whims of the ProDOS device manager. The parameters are passed in fixed absolute zero page locations, and device numbers are mapped from ProDOS conventional unit numbers to the sequential unit numbers required by the non extended SmartPort. All errors which the non extended SmartPort can return are mapped on ProDOS calls to the following error codes:

$27	I/O Error
$28	No Device Connected
$2B	Device is Write Protected
$2F	Device is Offline (No media)

Any 'fatal' error (bit 6 clear, bits 5-0 non-zero) that is not $28, $2B, or $2F is mapped to I/O Error ($27). All 'non-fatal' errors (bit 6 set; codes $50-$7F) is not considered an error and is mapped to $00.

Summary of Extended Commands and Parameter Lists

Command	Status	ReadBlock	WriteBlock	Format	Control	Init
CMDNUM	$40	$41	$42	$43	$44	$45
CMDLIST Byte 0:	$03	$03	$03	$01	$03	$01
1:	Unit #	Unit #	Unit #	Unit #	Unit #	Unit #
2:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
3:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
4:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
5:	StatList Ptr	Buffer Ptr	Buffer Ptr		CtrlList Ptr	
6:	StatusCode	Block Addr	Block Addr		Ctrl Code	
7:		Block Addr	Block Addr			
8:		Block Addr	Block Addr			
9:		Block Addr	Block Addr			
10:						
11:						

Summary of Extended Commands and Parameter Lists

Command	Open	Close	Read	Write		
CMDNUM	$46	$47	$48	$49		
CMDLIST Byte 0:	$01	$01	$04	$04		
1:	Unit #	Unit #	Unit #	Unit #		
2:			Buffer Ptr	Buffer Ptr		
3:			Buffer Ptr	Buffer Ptr		
4:			Buffer Ptr	Buffer Ptr		
5:			Buffer Ptr	Buffer Ptr		
6:			Byte Count	Byte Count		
7:			Byte Count	Byte Count		
8:			*	*		
9:			*	*		
10:			*	*		
11:			*	*		

* This parameter is device specific

Notes:
1) The read byte count and the Control call list contents cannot be larger than 767 bytes.
2) Upon return from the Read call, the byte count bytes will contain the number of bytes actually read from the device.

SUMMARY OF SMARTPORT ERROR CODES

Acc Value	Error Type	Description
\$00		No error
\$01	BADCMD	A nonexistent command was issued.
\$04	BADPCNT	Bad call parameter count. This error will occur only if the call parameter list was no properly constructed.
\$06	BUSERR	A communications error with the IWM occured.
\$11	BADUNIT	An invalid unit number was givien.
\$1F	NOINT	Interrupt devices not supported.
\$21	BADCTL	The control or status code is not supported by the device.
\$22	BADCTLPARM	The control list contains invalid information.
\$27	IOERROR	The device encountered an I/O error.
\$28	NODRIVE	The device is not connected. This can occur if the device is not connected but its controller is.
\$2B	NOWRITE	The device is write protected.
\$2D	BADBLOCK	The block number is not present on the device.
\$2F	OFFLINE	Device off line or no disk in drive.
\$30-\$3F	DEVSPEC	These are device specific error codes
\$40-\$4F	RESERVED	
\$50-\$5F	NONFATAL	A device specific 'soft' error. The operation completed successfully, but some exception condition was detected.