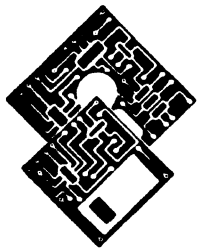


APPLE
PROGRAMMER'S
AND DEVELOPER'S
ASSOCIATION

290 SW 43rd. Street
Renton, WA 98055
206-251-6548

SCSI Development Package

Version 1.0

APDA#: KMSSDP

SCSI Development Package

Version 1.0

Developer Technical Support,
and
Macintosh Software Engineering,
Apple Computer, Inc.

This document is in manuscript form. It does not include

- ***final technical corrections***
- ***final editorial corrections***
- ***final art work***
- ***an index.***

Macintosh Plus / SCSI Developer Information

Introduction

This is a "starter kit" for developers of add-on SCSI (Small Computer Systems Interface) products for Macintosh. It is intended for those developers working on SCSI-based products; most developers do not need to know about SCSI (for instance, whether or not a particular volume is on an SCSI drive is hidden by the file system, and is generally of no interest to an application).

What you need for SCSI development

This section contains the following documents:

- "Macintosh SCSI Developer Information": This document, an overview of the development process for SCSI on Macintosh.
- "Example SCSI Blocked-Device Driver": a document describing a sample SCSI disk driver.
- "Patches to the SCSI Manager": a document covering known bugs in the ROM-based SCSI Manager software, and information on how we've fixed them.

Elsewhere in this package, you'll find:

- The SCSI Manager chapter of *Inside Macintosh Volume IV:* an overview of ROM-based software support for SCSI devices.
- On diskette: source code for the sample driver, and SCSI-related equates.

Other parts

Not included in this package is this other important information:

- Information about the proposed Small Computer System Interface standard (American National Standard Committee draft proposal ANSC X3T9.2/82-2).
- Information about the SCSI device you are connecting to Macintosh (in particular, information about the SCSI controller used by your device to manage communications on the SCSI bus).
- Of course: *Inside Macintosh*, "Outside Apple", and the Software Supplements!

How to proceed

It's important to have a working knowledge of how SCSI works, and how your device works; read the ANSC draft document and the manual that came with the SCSI controller you're using.

Once you understand SCSI and your device, you're ready to look into how SCSI relates to Macintosh. Read the SCSI Manager chapter of *Inside Macintosh*; it covers software in the Macintosh ROM used to manage SCSI communications. "Patches to the SCSI Manager" will explain several problems in the ROM-based code, along with how they can be avoided.

Study the sample driver source code and documentation. They show a number of "real-life" pitfalls (& workarounds) involved in adding a SCSI blocked device to Macintosh Plus.

You will probably find it helpful to acquire a logic analyzer to aid in your development; if you contact Technical Support to ask for help with a problem (like "I'm losing bytes!"), we'll ask what you saw when you looked for the problem with an analyzer!

A note about Licensing

The sample driver included in this package is **not** intended to be shipped as a product; it is therefore not licensable. It is provided to show how such a driver might be written (by someone with not a lot of time to tell developers how to write drivers! [see note below]).

You must still license the System and Finder if you want to ship them with a product; please contact Apple's Software Licensing department at (408) 973-4667 for more information.

A note about help from Developer Technical Support

The Developer Technical Support group is chartered to assist developers in creating Macintosh (and Apple II) products *for market*. Unfortunately, if you happen to have gotten a good deal on an old 370-megabyte cartridge drive that you want to hook up to your Macintosh, we can't really help you (there are only six of us for all Macintosh development, and four thousand of you!).

Whether or not you're working on a product for market, you may be able to get assistance from various independent developers groups, or from various consultants who specialize in Macintosh development (see the classified ads in the back of "Outside Apple", or write to us at the technical support address below and perhaps we can recommend somone).

If you are developing products for market, you should be a Certified Developer. It's free, and gets you mail or electronic-mail technical support, developer's discounts on Apple products, and a subscription to "Outside Apple". For more info, write to Apple Developer Relations, M/S 3-W, 20525 Mariani Avenue, Cupertino CA 95014.

If you're a Certified Developer and are developing a product and come upon a stumbling block, and this documentation doesn't help you over it, you can ask us for help. We answer technical questions from Certified Developers, via U.S. Mail (at the address below), via MCI Mail (mailbox "MACTECH"), or by telex (6502150798), generally with less than three days turnaround; however, please do your best to solve the problem on your own before asking us for help (also, please bear in mind that we're Macintosh experts, not really SCSI experts!).

If you have comments or suggestions about the enclosed materials, we'd like to hear them; please write to us at this address:

Apple Developer Technical Support: SCSI
20525 Mariani Avenue, M/S 3-T
Cupertino, CA 95014

Example SCSI Driver for Macintosh Plus

Introduction

This document describes an example SCSI blocked-device driver for Macintosh Plus.

This document and the accompanying software are directed at SCSI devices which can be used as disk drives by the Macintosh File Manager. No attention is paid to non-disk devices (such as archive tape drives, printers, etc); further specifications are planned for these kinds of devices (watch for news in future Software Supplements, and "Outside Apple"; if you have input to these specifications, please write to Apple Developer Technical Support, Attn: SCSI, 20525 Mariani Ave M/S 3-T, Cupertino, CA 95014).

This driver is an example to help get you started on your own driver; it is **not** intended as finished software. There are several "interesting" problems which must be overcome to get an SCSI blocked-device driver running on Macintosh; this software is intended to help you over these hurdles with a minimum of delay. Do not, however, assume that this software is bug-free; Apple Computer will not be responsible or liable for this software (or documentation) in any way. In fact; the previous version of this software (dated 26 March 1986) contained several errors which we didn't catch until after the package had shipped! (The ones we've found out about are fixed in this version, and noted at the bottom of this document. This version probably contains a new set of bugs!)

The example driver

The example driver grew out of one written by Macintosh Software Engineering to allow them to test the ROM-based SCSI Manager routines. It has been rewritten and commented to help third-party developers to get their SCSI devices running on Macintosh with a minimum of effort.

The driver was not written to be a clear example of how to write drivers; this driver attempts to show workarounds for the many problems which had to be dealt with in bringing up an SCSI hard disk, usually without explaining why all the workarounds are there.

Note that there are several desirable features missing from this driver; for instance, it only supports one non-partitioned non-ejectable volume per SCSI controller. If your device supports multiple drives on one SCSI controller, partitioning, or ejectable cartridges, you will want to add support for these features to this driver. In writing this driver, the emphasis was on Macintosh system issues, not on SCSI functionality; as a result, some areas of the driver's design have suffered (error handling, for instance).

A few notes about booting from an SCSI device

The process used to boot off of SCSI devices is described in the SCSI Manager chapter of *Inside Macintosh Volume IV*.

SCSI drivers are unique in that they are not in ROM and not in the System file. Normally, the ROM only knows about the ROM-based drivers at boot time. For SCSI, code was added to the ROM (we'll call it SCSIBoot, though it's not an externally- accessible routine) to allow drivers to be read in from disk before the system looks for volumes from which to boot. To keep the ROM simple, most of the work of loading an SCSI driver is left to the driver itself; that is, the code which is read in is assumed to be executable, and is given control so that it may install a driver in the system (in the example driver, you'll see that the first thing in the driver is not the normal driver header, but a branch to a routine which hooks the driver to the unit table, etc.).

The SCSIBoot code in the ROM makes no assumptions about the code it loads and transfers control to. SCSIBoot itself is called by the normal boot code, which uses the following sequence to boot Macintosh:

- Call SCSIBoot to find SCSI devices which might be bootable
- Start at the beginning of the drive queue

- If there's another drive there:
  - Try to read the boot blocks from it.
  - If the boot blocks are valid:
    - Try to mount the volume on that drive... etc.
  - If we can't read the boot blocks, or mount, or can't find System, etc:
    - Give the driver an "eject" control call.
    - If the driver returns an error, assume it's non-ejectable: don't try to boot from it next time through the drive queue.
- If we haven't succeeded in finding a bootable drive by the time we run out of drives in the drive queue, start again at the top of this list.

The boot code remembers non-ejectable drives (from which it has failed to boot) as bits in a word. This means that drive numbers should be in the range 1..15 (because of built-in drivers, SCSI drivers should start looking for free drive numbers at drive number 5; see the note in the source).

Before running through all SCSI devices, the SCSIBoot code issues a _SCSIReset call to reset the SCSI bus. However, some (newer) SCSI controllers go into a "unit attention" state upon reset; because the ROM expects to be able to read from the device immediately after reset, *any such device cannot be a Macintosh Plus startup volume* (because the SCSIBoot code would not be able to successfully read the boot blocks from it). Use controllers without this "unit attention" feature with Macintosh Plus.

It does no good to post a disk-inserted event for an SCSI drive at driver-install time; the ROM will try to boot from your drive if it finds your drive in the drive queue. In fact, any drive it doesn't boot from is forgotten, so we have to pull a few special tricks to "remind" the system if our drive isn't bootable (again, refer to the source).

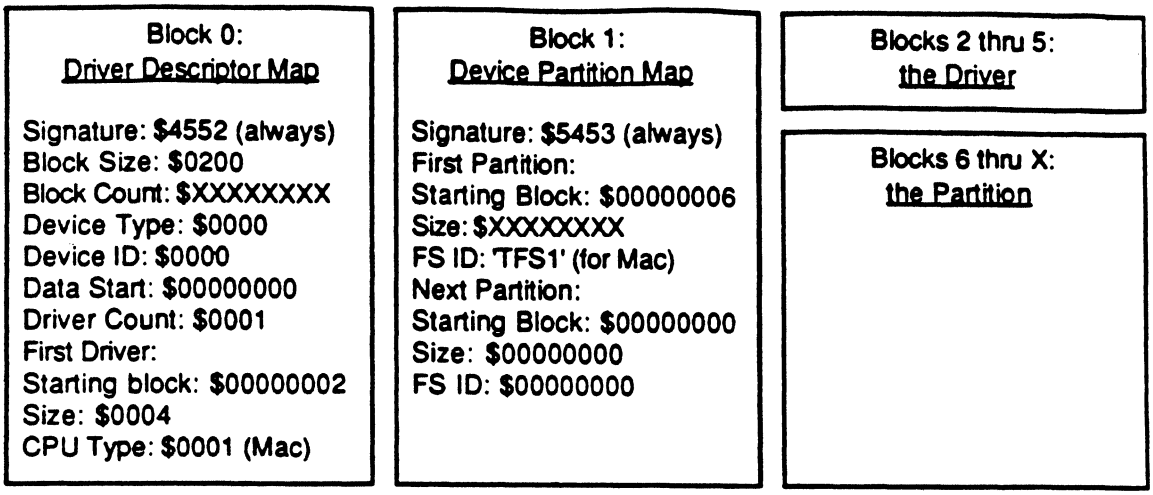
The first drive found (in the drive queue) with a System file and Finder is used as the startup volume. Once mounted, an SCSI-based volume looks like any other volume to the system, and no special handling is needed.

If no bootable drive is found on the first pass through the drive queue, the boot code starts over with SCSIBoot (which means that the SCSI bus may be reset again after your device's driver has been loaded!).

Making your drive bootable

You must place your driver on your hard disk for it to be bootable. If your drive cannot be read (by the ROM's generic read routines) immediately after the SCSI bus is reset, then it cannot be bootable (this is the case with most controllers which support "unit attention", described above).

The SCSI Manager chapter of *Inside Macintosh Volume IV* contains descriptions of the tables on blocks 0 and 1 of a bootable SCSI device (equates for these tables are now in the Software Supplement equate files). These tables were designed to allow several CPUs' operating systems to own their own partitions on a drive; if you only support one operating system (Macintosh HFS), your drive will look something like this (the size of your driver may be different):



The utility which you distribute with your drive should do (at least) the following, to prepare a drive for use as a Macintosh blocked volume:

- Take care of any low-level formatting required by the drive (perhaps reading defect maps from files on a floppy distributed with the drive).
- Set up and write out the Driver Descriptor Map, the Device Partition Map, and the driver, so that the system will be able to boot from the device.
- Open the driver from floppy (you must simulate the process that the system uses at boot time), then call DIZero to initialize the partition as a volume.

Your utility should also take care of any routine maintenance, such as partition management (if you decide to support multiple partitions or operating systems), driver updating, mounting of a drive after boot time, etc.

The sample driver contains functionality to support an installation utility; this way, the driver can be used by the utility to write itself (as well as the DDM and DPM) out to the newly formatted disk.

A utility can install the sample driver by JSRing to the driver code; the first instruction is a branch to the install routine within the driver (the driver cannot be a 'DRVR' resource because of this branch instruction). After the install routine has been called, your utility will need to post a disk-inserted event for the drive whose drive queue element was created by the install routine (your utility can find this at an "agreed-upon" location in the driver's globals).

Building the software

The example driver was written using a pre-release version of the Macintosh Programmer's Workshop Assembler, part of Apple's new Macintosh-based development environment. If you use a different assembler, you will probably have to change some of the source files (mainly the pseudo-opcodes and macro calls). MPW is not yet available as of this writing; please do not call us to ask for it (the conversions to other currently-available development environments are not difficult!).

This driver requires the SCSI equates source file, included in this Supplement.

Known bugs

There are a few known problems with this software (if you find more, please write and let us know, at the above address).

- SCSI error handling should be improved. Currently, after an error is detected, the SCSI manager calls _SCSIComplete to clean up the bus, then returns IOErr. It should request sense status from the drive, and handle errors appropriately.

- The Install code is a large percentage of the driver code, and the memory it occupies could be purged after installation time by resizing the memory block that the code resides in, or by keeping the actual driver as a separate area on the disk, and having the install code load it in separately.

Bugs fixed since the 26-March-86 version

Here's a summary of the changes made since the last version; if that version influenced your driver, you should make sure you didn't copy my mistakes!

Thanks to all of you who reported these bugs:

- The hard-coded value for the new _GetTrapAddress call in SetBlindOK was incorrect (let this be a lesson in using equated values!).

- Also in SetBlindOK, a CLR.W instruction was used to clear TickleFlag (a byte value).

- My driver source, my documentation, and the alpha draft of the SCSI Manager chapter of *Inside Macintosh Volume IV* differed on the proper value for the signature word for one of the device tables; the driver was correct.

- The first version of this driver contained support for both 256- and 512-byte physical sectors. This was interesting as a feature, but made the example cloudy (and besides, most popular controllers can be configured for 512-byte sectors). In removing this feature at the last minute, I incorrectly recoded an instruction in setting up the transfer information block for the transfer (in the routine DiskPrime); rather than moving the sector size into the low word of the scNoInc instruction, I did a long move into the low word (a bad booboo).

- In my own hacked-up installer, I used the driver to write itself onto the SCSI device after formatting. Since the device I was accessing didn't have a volume mounted (and since the area I was writing wasn't within the partition anyway!), I added a special control call. Later, I changed the installer to no longer need this control call. At some point after that, I had to retype a large section of code from a listing, and made a typo in a register reference (in DoSCSICmd).

- In converting to the "real" SCSI equates file, several labels had to be changed (for instance, in the driver globals, SCSICmd [the SCSI command block] was changed to SCmd to avoid a conflict).

Patches to the SCSI Manager in Macintosh Plus

Introduction

There are a number of bugs in the ROM version of the Macintosh Plus SCSI Manager; these bugs have been fixed by patching the _SCSIDispatch trap in RAM.

A number of developers received a file (variously called "SCSI Helper", "SCSI Items", or "Preliminary SCSI Helper") containing these patches (this file was included in the 26 March 1986 release of this package).

The newest System file (version 3.2, included in this Software Supplement) contains these patches, so the SCSI Helper file is no longer necessary; in fact, since the patches in the System file would be replaced by the [outdated] SCSI Helper patches, it would be a good idea to throw away any versions of SCSI Helper you may have.

Furthermore, you should tell your users to always use this System file (or a more recent one) with your SCSI products.

About the fixes

These patches fix several problems in the Macintosh Plus ROM implementation of the SCSI Manager:

- Blind writes (_SCSIWBlind) were known to fail on most SCSI controllers. They may still fail – test them on your controller carefully. Note that blind transfers should only be used when transferring disk blocks (ie, not when transferring the data for a Mode Select command, for instance) because your controller may not be able to keep up the transfer rate if it's interpreting the data as it receives it (as it may when transferring non-disk-block data).

- Blind reads (_SCSIRBlind) were known to fail on a few controllers. They should now work for all controllers. See note above about blind transfers in general.

- A bug in the ROM version of _SCSIStat (the routine which returned the NCR chip's status byte) prevented it from ever working. SCSIHelper contains a fixed version.

Your driver must test for the presence of the patches at runtime; furthermore, you must be able to boot from your device without benefit of the patches (because, of course, they're not going to be loaded until after the System has been loaded). You should also be able to run properly (even if slowly, without blind transfers) if the patches are not present, in case the user happens to boot from a non-3.2 System.

To test for the presence of the patch at runtime, you should add a flag to your driver's (DCtlStorage) local variables (call it BlindOK), and add the following code to your driver's Open routine:

```
;Initialize the "blind-transfer" flag in our globals,
;and set us up for a NeedTime call after we've finished booting.
;(assumes that a pointer to our globals is in A0, and a pointer
;to our DCE is in A1).
          MOVE.L  A1,-(SP)                ;(save A1 across call)
          BSR     SetBlindOK              ;set up the BlindOK flag
          MOVE.L  (SP)+,A1                ;(restore A1)
          BEQ.S   @1                      ;all patched!
          MOVE.W  #1,DCtlDelay(A1)        ;we'll need time later --
          BSET    #dNeedTime,DCtlFlags(A1) ;flag it
```

Your driver's Control routine will eventually get an accRun call (CSCode = accRun). This will happen as soon as the first application calls SystemTask. When this happens, BSR to the following

routine (before exiting through IODone in the normal manner):

```
;We've gotten an accRun call. Check to see if it's OK to do blind
;transfers now, then reset the dNeedTime flag in our DCE.
;(assumes that a pointer to our globals is in A0, and a pointer
;to our DCE is in A1).
GotNeedTime
            CLR.W   DCtlDelay(A1)              ;we don't need time anymore
            BCLR    #dNeedTime,DCtlFlags(A1)   ;unflag it
            BSR     SetBlindOK                 ;re-set up the BlindOK flag
            RTS
```

The following routine checks to see if it's OK to do blind transfers; place it somewhere in your driver (it's called from the added Open and Control routines above):

```
;Set up the "blind-transfer" flag in our globals, based on ROM
;version and patch state of the _SCSIDispatch trap. (This check
;for ROM version only applies here -- do not use it for general
;version checking).
;
;(This routine assumes that a pointer to our globals is in A0)
SetBlindOK MOVE.L  A3,-(SP)                    ;save A3
           MOVE.L  A0,A3                       ;keep globals ptr here
           CLR.B   BlindOK(A3)                 ;default to "not OK!"
           MOVE.L  ROMBase,A0                  ;get pointer to ROM start
           MOVE.W  8(A0),D0                    ;get ROM version word
           CMP.W   #$0075,D0                   ;are we newer than MacPlus?
           BNE.S   @1                          ;yes! Make blind xfers OK.

           ;we're running on Macintosh Plus ROMS -- has _SCSIDispatch
           ;been patched in RAM? (Since _SCSIDispatch is a new trap, we
           ;have to use the new flavor of GetTrapAddress).
           ;
           ;Note: The last version of this example had the _GetTrapAddress
           ;trap incorrect: the correct version is shown.
           MOVE.W  #$A815,D0                   ;get a SCSIDispatch trap word
           DC.W    $A346                       ;_GetTrapAddress for new OS
           CMPA.L  ROMBase,A0                  ;is it below ROMbase?
           BGE.S   @2                          ;no: Unpatched, so no blinds.

;Either the ROMs are newer than Macintosh Plus's, or we've patched
;SCSIDispatch in RAM. Either way, blind transfers should be OK now.
@1         ST      BlindOK(A3)

;Either we just made blind transfers OK, or it's not OK to use them.
@2         MOVE.L  (SP)+,A3                    ;restore A3
           TST.B   BlindOK(A3)                 ;set CCR flags
           RTS                                 ;and exit.
```

Anytime you use _SCSIRBlind to transfer disk block data, test the flag; if it's clear, use _SCSIRead instead (the same goes for _SCSIWBlind if you are using it).