

Copies of this proposal may be purchased from:  
Global Engineering, 2805 McGaw St, Irvine, CA 92714  
800-854-7179 714-261-1455

BSR X3.\*\*\*  
X3T9.2/90-186

An electronic copy of this document is available from the SCSI Bulletin Board (316-636-8700).

This document has been prepared according to the style guide of the ISO (International Organization of Standards).

working draft proposed American National  
Standard for Information Systems -

SCSI-2 Common Access Method

Transport  
and

SCSI Interface Module

Rev 2.3 February 25, 1991

If this document was printed in a 2-up form directly from the printer, NOTES had to be adjusted to fit into a half-page, which may have resulted in an imperfect representation of the format within the NOTE. This is most likely to occur if a series of NOTES are mixed in without any line separation.

**This revision contains recommended changes by Jerry Armstrong and the DOS OSD working group. You will find that the DOS OSD has been extensively revised.**

**John Gallant also submitted a list of change requests, most of which are included. Please look them over carefully, and be sure to advise Dal by fax at 408-867-2115 of any objections.**

Secretariat

Computer and Business Equipment Manufacturers Association (CBEMA)

Abstract: This standard defines the software interface between device drivers and the Host Bus Adapters or other means by which SCSI peripherals are attached to a host processor. The software interface defined provides a common interface specification for systems manufacturers, system integrators, controller manufacturers, and suppliers of intelligent peripherals.

**This is an internal working document of X3T9.2, a Task Group of Accredited Standards Committee X3. As such this is not a completed standard. The contents are actively being modified by the X3T9.2 Task Group. This document is made available for review and comment only.**

POINTS OF CONTACT:

John B. Lohmeyer  
Chairman X3T9.2  
NCR  
3718 N Rock Rd  
Wichita KS 67226

316-636-8703

I. Dal Allan  
Vice-Chairman X3T9.2  
ENDL  
14426 Black Walnut Court  
Saratoga CA 95070

408-867-6630

59

Foreword (This Foreword is not part of American National Standard X3.\*\*\*-199x.)

In this standard, the Transport (XPT) and SCSI Interface Module (SIM) for the SCSI-2 Common Access Method is defined.

When the Small Computer System Interface (SCSI) was introduced, a large number of systems integrators included support in their operating systems. However, they were parochial in implementation and a diverse set of strategies to support SCSI devices were implemented in software.

Some companies published their specifications and encouraged third-party suppliers to add new peripherals. Others failed to add support for SCSI or did not publish the specifications. An increasing demand developed for some common method to attach SCSI peripherals to a number of operating systems and a large range of computer systems. Much of this impetus stemmed from the growth in the desktop computing environment.

In October 1988 a number of peripheral suppliers formed the Common Access Method Committee to encourage an industry-wide effort to adopt a common software interface to despatch input/output requests to SCSI peripherals.

The primary objective was to define a set of software constructs and tables that would permit the manufacturers of host adapters to provide software or microcode to interpret requests in a common manner.

Out of the proposals made by a large number of contributors, the CAM Committee selected the best concepts and used them to develop the standard.

Some of the companies which contributed had designed their own methods to support SCSI devices, and for the most part set aside individual business considerations to foster the development and adoption of this standard.

Suggestions for improvement of this standard will be welcome. They should be sent to the Computer and Business Equipment Manufacturers Association, 311 First Street N.W., Suite 500, Washington, DC 20001.

This standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Information Processing Systems, X3. Committee approval of this standard does not necessarily imply that all committee members voted for its approval. At the time it approved this standard, the X3 Committee had the following members:

X3 Committee list goes here:

Subcommittee X3T9 on I/O interfaces, which reviewed this standard, had the following members:

X3T9 Committee list goes here:

Task Group X3T9.2 on Lower-Level Interfaces, which completed the development of this standard, had the following members:

X3T9.2 Committee list goes here:

The initial development work on this standard was done by the CAM Committee, an industry group formed for this purpose. The membership of the CAM Committee consisted of the following organizations:

Adaptec	Data Technology	NCR
AMD	Eastman Kodak	Olivetti
Apple	Emulex	Quantum
AT&T Bell Labs	Fujitsu uElectronics	Scientific Micro Systems
Caliper	Future Domain	Seagate
Cambrian Systems	Hewlett Packard	Sony
Cipher Data	IBM	Storage Dimensions
Cirrus Logic	Imprimis	Sun Microsystems
Columbia Data	Interactive Systems	Syquest Technology
CompuAdd	JVC	Sytron
Conner Peripherals	LMS OSD	Trantor
Dell Computer	Maxtor	Western Digital
Digital Equipment	Micropolis	
DPT	Miniscribe	

TABLE OF CONTENTS

1.	Scope	1	7.1.7	SIM Interface	18
1.1	Description of Clauses	1	7.2	Novell Operating System	19
2.	References	2	7.2.1	Initialization	19
3.	General Description	2	7.2.2	De-Registration	20
3.1	Environment	2	7.2.3	Accessing the XPT	20
3.2	Peripheral Driver Functions	3	7.2.4	Hardware Registration	21
3.3	XPT Functions	4	7.2.5	Miscellaneous	21
3.4	SIM Functions	4	7.3	DOS (Disk Operating System)	21
4.	Definitions and Conventions	4	7.3.1	Initialization	21
4.1	Definitions	4	7.3.1.1	Multiple XPTs	22
4.2	Conventions	5	7.3.1.2	Device Table Handling	22
5.	Background	6	7.3.2	Accessing the XPT	22
5.1	Software	6	7.3.2.1	Testing for the presence of the XPT/SIM	22
5.2	CAM (Common Access Method)	6	7.3.2.2	Sending a CCB to the XPT	23
5.2.1	XPT (Transport)	6	7.3.3	Callback on Completion	24
5.2.2	SIM (SCSI Interface Module)	6	7.3.4	Asynchronous Callbacks	24
5.2.3	CCB (CAM Control Block)	7	7.3.5	Pointer Definition	25
5.2.4	OSD (Operating System Dependent)	7	7.4	OS/2 (Operating System 2)	25
5.3	Principles of Operation	7	8.	CAM Control Blocks	25
5.4	Requirements	8	8.1	CCB Header	26
6.	Transport	8	8.1.1	CAM Control Block Length	26
6.1	Accessing the XPT	8	8.1.2	XPT Function Code	26
6.2	Initialization	9	8.1.3	CAM Status	27
6.3	Callback on Completion	9	8.1.4	Path ID	27
6.4	SCSI Request Queues	9	8.1.5	CAM Flags	27
6.4.1	The Target/LUN and the Peripheral Driver	10	8.2	Function Codes	27
6.4.2	The SIM	10	8.2.1	Get Device Type	27
6.4.3	SIM Queuing	10	8.2.2	Path Inquiry	28
6.4.3.1	SIM Queue Priority	10	8.2.3	Release SIM Queue	31
6.4.3.2	Tag Recognition	10	8.2.4	Set Async Callback	31
6.4.3.3	Error conditions and Queues within the Subsystem	10	8.2.5	Set Device Type	32
6.5	SIM Handling of SCSI Resets	11	8.3	SCSI Control Functions	33
6.6	Asynchronous Callback	12	8.3.1	Abort XPT Request	33
6.7	Autosense	13	8.3.2	Reset SCSI Bus	34
6.8	Loadable Modules	14	8.3.3	Reset SCSI Device	34
7.	OSD (Operating System Dependent) Operation	15	8.3.4	Terminate I/O Process Request	35
7.1	UNIX Operating System	15	9.	Execute SCSI I/O	36
7.1.1	Initialization	15	9.1	CAM Control Block to Request I/O	36
7.1.2	Accessing the XPT	16	9.1.1	Address of this CCB	37
7.1.2.1	From the Peripheral Driver	16	9.1.2	Callback on Completion	37
7.1.2.2	From the SIM	16	9.1.3	CAM Control Block Length	37
7.1.3	Callback on Completion	17	9.1.4	CAM Flags	37
7.1.4	Pointer Definition in the UNIX Environment	17	9.1.4.1	Byte 1 Bits	38
7.1.5	Request Mapping Information	17	9.1.4.2	Byte 2 Bits	39
7.1.6	XPT Interface	17	9.1.4.3	Byte 3 Bits	39
7.1.6.1	Functions for Peripheral Driver Support	17	9.1.4.4	Byte 4 Bits	40
7.1.6.2	Functions for SIM Module Support	18			

19

9.1.5	CAM Status	40
9.1.6	CDB	42
9.1.7	CDB Length	42
9.1.8	Data Transfer Length	42
9.1.9	Function Code	43
9.1.10	LUN	43
9.1.11	Message Buffer Length (Target-only)	43
9.1.12	Message Buffer Pointer (Target-only)	43
9.1.13	Next CCB Pointer	43
9.1.14	Number of Scatter/Gather entries	43
9.1.15	Path ID	43
9.1.16	Peripheral Driver Pointer	43
9.1.17	Private Data	43
9.1.18	Request Mapping Information (OSD)	43
9.1.19	Residual Length	44
9.1.20	SCSI Status	44
9.1.21	Sense Info Buffer Length	44
9.1.22	Sense Info Buffer Pointer	44
9.1.23	SG List/Data Buffer Pointer	44
9.1.24	Tagged Queue Action	44
9.1.25	Target ID	44
9.1.26	Timeout Value	44
9.1.27	VU Flags	45
9.2	Command Linking	45
10.	Target Mode (Optional)	45
10.1	Enable LUN	46
10.2	Phase Cognizant Mode	48
10.2.1	Target Operation of the HBA	48
10.2.2	Execute Target I/O	49
10.3	Processor Mode	50
10.3.1	CCB Acceptance	50
10.3.2	Target Operation of the HBA	50
11.	HBA Engines	51
11.1	Engine Inquiry	51
11.2	Execute Engine Request (Optional)	52

FIGURES

FIGURE 3-1	CAM ENVIRONMENT MODEL	3
------------	-----------------------	---

TABLES

TABLE 6-1	ASYNC CALLBACK OPCODE DATA REQUIREMENTS	13
TABLE 8-1	CAM CONTROL BLOCK HEADER	25
TABLE 8-2	SUPPORT OF SCSI MESSAGES	26
TABLE 8-3	XPT FUNCTION CODES	27
TABLE 8-4	GET DEVICE TYPE CCB	28
TABLE 8-5	PATH INQUIRY CCB - Part 1 of 2	29
TABLE 8-5	PATH INQUIRY CCB - Part 2 of 2	30
TABLE 8-6	RELEASE SIM QUEUE	31
TABLE 8-7	SET ASYNC CALLBACK CCB	32
TABLE 8-8	SET DEVICE TYPE CCB	33
TABLE 8-9	ABORT XPT REQUEST CCB	33

TABLE 8-10	RESET SCSI BUS CCB	34
TABLE 8-11	RESET SCSI DEVICE CCB	35
TABLE 8-12	TERMINATE I/O PROCESS REQUEST CCB	35
TABLE 9-1	SCSI I/O REQUEST CCB	36
TABLE 9-2	CAM FLAGS (OSD)	38
TABLE 9-3	SCATTER GATHER LIST	39
TABLE 9-4	CAM STATUS	41
TABLE 10-1	ENABLE LUN CCB	46
TABLE 10-2	TARGET CCB LIST	46
TABLE 11-1	ENGINE INQUIRY CCB	52
TABLE 11-1	EXECUTE ENGINE REQUEST CCB	53

Information Processing Systems --

Common Access Method --

SCSI and Generic I/O

## 1. Scope

This standard defines the CAM (Common Access Method) for SCSI (Small Computer Systems Interface).

The purpose of this standard is to define a method whereby multiple environments may adopt a common procedure for the support of SCSI devices.

The CAM provides a structured method for supporting peripherals with the software (e.g. device driver) and hardware (e.g. host bus adapter) associated with any computer.

SCSI has provided a diverse range of peripherals for attachment to a wide range of computing equipment. Some system manufacturers have developed approaches for SCSI attachment which are widely followed, increasing the applications available for the attachment of SCSI peripherals. In markets where no standard method of attachment exists, however, variations between third party sellers has made it near-impossible for end users to rely on being able to attach more than one SCSI peripheral to one host bus adapter.

In an effort to broaden the application base for SCSI peripherals an ad hoc industry group of companies representing system integrators, controllers, peripherals, and semiconductors decided to address the issues involved.

The CAM Committee was formed in October, 1988 and the first working document of the XPT/SIM for SCSI I/O was introduced in October, 1989.

### 1.1 Description of Clauses

Clause 1 contains the Scope and Purpose.

Clause 2 contains Referenced and Related International Standards.

Clause 3 contains the General Description.

Clause 4 contains the Glossary.

Clause 5 describes the services provided by the XPT and SIM.

Clause 6 describes the facilities that use the Transport and SIM.

Clause 7 describes the ways that the Operating Systems support CAM and access the XPT.

Clause 8 contains the description of non-I/O functions supported by the XPT and SIM.

Clause 9 contains the description of I/O functions supported by the XPT and SIM.

Clause 10 contains the description of Target Mode functions supported by the

XPT and SIM.

## 2. References

ISO DIS 10288 (ANSI X3.131-1990)  
SCSI-2, Enhanced Small Computer Systems Interface

## 3. General Description

The application environment for CAM is any computer addressing a SCSI peripheral through a protocol chip on a motherboard, an ATA interface, or a Host Bus Adapter.

SCSI is a widely-used interface which provides common attachment for a variety of peripherals. Unfortunately, there is no common way to provide access to SCSI peripherals.

The purpose of the Common Access Method is to define a standard for the support of Host Bus Adapters and the like by device driver software.

Software in the Operating System dispatches I/O (Input/Output) requests to the SCSI peripherals in a number of different ways depending on the software architecture. The OSD (Operating System Dependencies) are defined in Clause 6 for named software and hardware platforms.

### 3.1 Environment

A model of the CAM usage environment is illustrated in Figure 3-1, where there may be multiple application and several device drivers attached to support the peripherals on the system.

Requests for SCSI I/O are made through the CAM Transport (XPT) interface. The XPT may execute them directly or pass them on to a lower level SIM for execution.

The XPT (Transport) function is illustrated as a separate element. In many applications, the XPT operations will be incorporated into a single module which integrates both XPT and SIM functionality. The logical separation between the two is maintained as there may be more than one SIM loaded.

A separate routing service may be provided by the operating system or the routing function can be achieved through chaining when multiple SIMs are loaded.

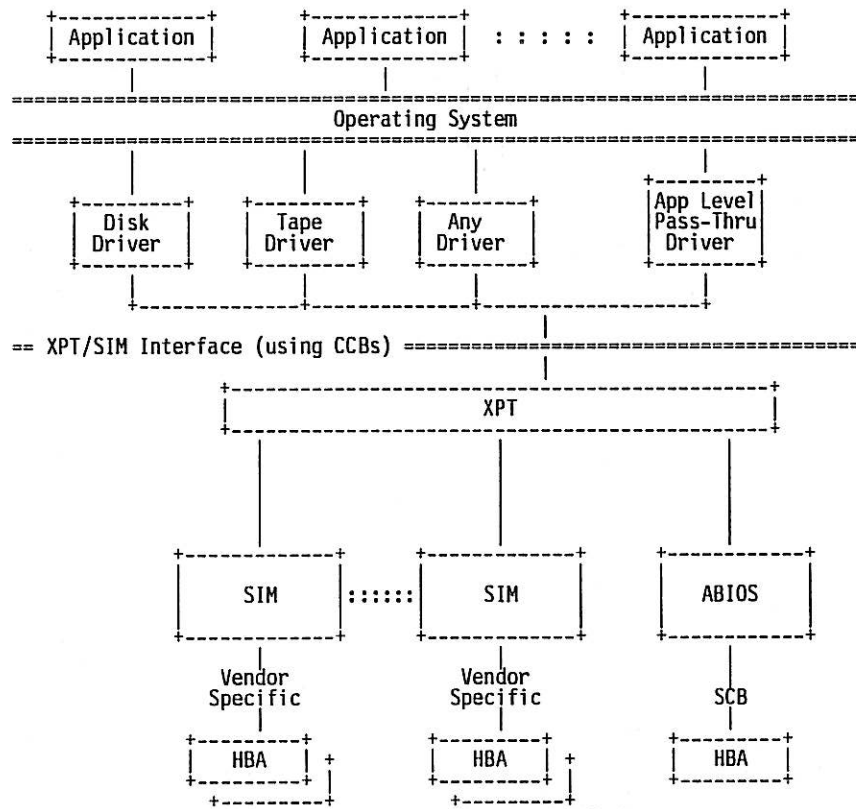


FIGURE 3-1 CAM ENVIRONMENT MODEL

### 3.2 Peripheral Driver Functions

Peripheral drivers provide the following functionality:

- a) Interpretation of application or system level requests.
- b) Mapping of application level requests to XPT/SIM Control Blocks.
- c) Requesting of resources to initiate a CAM request.
  - CAM Control Blocks and supporting blocks that may be needed.
  - Buffer requirements.
- d) Handling of exception conditions not managed transparently by SCSI e.g. Check Condition status, unexpected Bus Free, Resets etc).
- e) Logging of exception conditions for maintenance analysis programs.
- f) Format utility or services required by format utilities.
- g) Establish parameters for HBA operation.
- h) Set up routing of SCSI requests to the correct Path/Bus, target and LUN.
- i) Initialization and configuration functions of a target not handled by a

- j) utility at installation and formatting time.
- j) Establish a timeout value for a task and pass this value in the CCB.

### 3.3 XPT Functions

XPT services provide the following functionality to process CCBs:

- a) Routing of the target CCB to the proper SIM.
- b) OSD allocation of CCB resources e.g. Get CCB, Free CCB.
- c) Maintenance of the SCSI Device Table. This consists of owning the table and servicing requests to read and write the table.
- d) Providing properly formatted control blocks and priming the fields needed to accomplish a request.
- e) Routing of asynchronous events back to peripheral driver.

### 3.4 SIM Functions

SIM services provide the following functionality to process CCBs:

- a) Perform all interface functions to the SCSI HBA.
- b) Manage or delegate, as required, all the SCSI HBA protocol steps.
- c) Distinguish abnormal behavior and perform error recovery, as required.
- d) Management of data transfer path hardware, including DMA circuitry and address mapping, and establish DMA resource requests (if necessary).
- e) Queueing of multiple operations for different LUNs as well as the same LUN and assign tags for Tag Queueing (if supported).
- f) Freeze and unfreeze the queue as necessary to accomplish queue recovery.
- g) Assuring that the completed operation is posted back to the initiating device driver.
- h) Management of selection, disconnection, reconnection, and data pointers of the SCSI HBA protocol.
- i) Mechanisms to accept the selecting and sensing of the SCSI HBA functions supported.
- j) Implement a timer mechanism, using values provided by the peripheral driver.

## 4. Definitions and Conventions

### 4.1 Definitions

For the purpose of this standard the following definitions apply:

- 4.1.1 Block: This defines an action to prevent access e.g. Busy.
- 4.1.2 CCB (CAM Control Block): The data structure provided by peripheral drivers to the XPT to control execution of a function by the SIM.
- 4.1.3 CDB (Command Descriptor Block): A block of information containing the SCSI opcode, parameters, and control bits for that operation.
- 4.1.4 DMA (Direct Memory Access): A means of data transfer between peripheral and host memory without processor intervention.
- 4.1.5 Freeze: This defines a software action to quiesce activity e.g. freeze the queue.
- 4.1.6 HBA (Host Bus Adapter): The hardware and microcode which provides the

interface between system memory and the SCSI bus.

4.1.7 Lock: This defines a hardware action e.g. data cartridge in a removable media drive.

4.1.8 Nexus: A block of information containing the SCSI device, LUN, and Queue Tag Number (if any, as used in command queuing).

4.1.9 Null: A value which indicates that the contents of a field have no meaning. This value is typically, though not necessarily, zero.

4.1.10 Optional: This term describes features which are not required by the standard. However, if any feature defined by the standard is implemented, it shall be done in the same way as defined by the standard. Describing a feature as optional in the text is done to assist the reader. If there is a conflict between text and tables on a feature described as optional, the table shall be accepted as being correct.

4.1.11 Reserved: Where this term is used for bits, bytes, fields and code values; the bits, bytes, fields and code values are set aside for future standardization. The default value shall be zero. The originator is required to define a Reserved field or bit as zero, but the receiver should not check Reserved fields or bits for zero.

4.1.12 SCB (Subsystem Control Block): The term defined by IBM to refer to an architecture to support SCSI Host Adapters.

4.1.13 SCSI (Small Computer Systems Interface): The I/O interface which this standard is designed to support.

4.1.14 SIM (SCSI Interface Module): A module designed to accept the CAM Control Blocks routed through the XPT in order to execute SCSI commands.

4.1.15 VU (Vendor Unique): This term is used to describe bits, bytes, fields, code values and features which are not described in this standard, and may be used in a way that varies between vendors.

4.1.16 XPT (Transport): A layer of software which peripheral drivers use to originate the execution of CAM functions.

#### 4.2 Conventions

Within the tables, there is a Direction bit which indicates In or Out. The presumption is from the view of the peripheral driver i.e. information is Out to the SIM from the peripheral driver and In to the peripheral driver from the SIM.

Certain terms used herein are the proper names of signals. These are printed in uppercase to avoid possible confusion with other uses of the same words; e.g., ATTENTION. Any lower-case uses of these words have the normal American-English meaning.

A number of conditions, commands, sequence parameters, events, English text, states or similar terms are printed with the first letter of each word in uppercase and the rest lower-case; e.g., In, Out, Request Status. Any lower-case uses of these words have the normal American-English meaning.

The American convention of numbering is used i.e., the thousands and higher multiples are separated by a comma and a period is used as the decimal point. This is equivalent to the ISO convention of a space and comma.

American:	0.6	ISO:	0,6
	1,000		1 000
	1,323,462.9		1 323 462,9

#### 5. Background

SCSI (Small Computer Systems Interface) is a peripheral interface designed to permit a wide variety of devices to coexist. These peripherals are typically, but not necessarily, attached to the host by a single SCSI HBA (Host Bus Adapter).

##### 5.1 Software

OS (Operating System) support for peripheral devices is normally achieved through peripheral drivers or utility programs. No single driver or program can reasonably support all possible SCSI peripherals, so separate drivers are needed for each class of installed SCSI device. These drivers need to be able to share the SCSI HBA hardware.

These drivers also have to work with a broad range of HBA hardware, from highly intelligent coprocessors to the most primitive, including a SCSI chip on a motherboard.

A standard SCSI programming interface layer is essential to insulate SCSI peripheral drivers and utilities from the HBA hardware implementation, and to allow multiple drivers to share a single SCSI hardware interface.

##### 5.2 CAM (Common Access Method)

This standard describes the general definition of the CAM (Common Access Method). CAM functionality has been separated into a few major elements.

###### 5.2.1 XPT (Transport)

The XPT (Transport) defines a protocol for SCSI peripheral drivers and programs to submit I/O requests to the HBA specific SIM module(s). Routing of requests to the correct HBA and posting the results of a request back to the driver are capabilities of the Transport.

###### 5.2.2 SIM (SCSI Interface Module)

The SIM (SCSI Interface Module) manages HBA resources and provides a hardware-independent interface for SCSI applications and drivers i.e. the SIM is responsible to process and execute SCSI requests, and manage the interface to the HBA hardware.

There are no requirements on how the SIM is implemented, in RAM (Random Access Memory) or ROM (Read Only Memory), provided the XPT is properly supported. A ROM-based SIM may need a transparent (to the user) software layer to match the SIM-required services to the specific manner in which they are requested of the OS.

65

### 5.2.3 CCB (CAM Control Block)

The CAM Control Block is a data structure passed from the peripheral driver to the XPT. The contents of the data structure describe the action required and provides the fields necessary for successful processing of a request.

### 5.2.4 OSD (Operating System Dependent)

The system environment in which the CAM is operating is a function of the hardware platform and the Operating System being executed e.g. the byte ordering is different between an Intel-based and a Motorola-based machine, and the calling structure differs greatly between Operating Systems.

Although the fields of a CCB may have a common meaning, the contents will vary by platform and OS. These dependencies cause differences in operation and implementation, but do not prevent interoperation on the same platform of two CAM modules implemented by different manufacturers.

The OSD issues are predominantly described in the XPT for each OS environment.

### 5.3 Principles of Operation

Ideally, a single XPT model would suffice for all OS environments for a single HBA, but this is impractical in light of the wide architectural differences between the various processor architectures.

Programming effort has been minimized by making the interfaces as similar as possible across OS platforms, and customizing the SIM for each HBA to maximize performance under each OS. HBAs vary widely in the capability and functions they provide so there may be an internal (transparent) interface to isolate hardware interface routines from routines which make use of OS resources.

In order to prevent each peripheral driver from having to scan the SCSI bus for devices at initialization, the XPT determines all installed SCSI devices and constructs an internal table. A XPT function is used by drivers and programs to access this table.

Peripheral drivers need to be developed with documentation provided by the operating system vendor in addition to that supplied by this standard.

Under Unix, the XPT and SIM would typically be compiled with the kernel at System Generation time, so that entry points would be resolved during linkage-editing.

Third party attachments may be supported without the need for a sysgen if suitable routing facilities are provided by the system vendor.

Under Novell, the XPT is supplied by Novell, and the SIM is implemented according to Novell documentation guidelines.

Under DOS, there is one logical XPT with one entry point, but it may consist of a number of separate modules (perhaps supplied for each HBA in the system).

Routing is a mechanism to support concurrent SIM modules being co-resident so that different HBAs can be mixed in the same system. This may be handled by

chaining the XPT entry points, defining additional character devices, or by a specific routing entity.

Once the SIM is loaded, the peripheral drivers integrate each type of SCSI device into the OS through XPT, independent of the installed HBA hardware.

Under OS/2 the equivalent to a XPT function is supplied by Microsoft, and the SIM is implemented according to Microsoft LADDR documentation guidelines.

### 5.4 Requirements

System requirements addressed in defining the CAM include:

- a) Device drivers and programs should be able to use any SCSI command, both defined in SCSI-2 X3.131-1990 or Vendor Unique.
- b) No assumptions on the size and format of transferred data.
- c) Allowing all the capabilities of high end host adapters to be fully utilized and accommodate HBAs which do most of the SCSI processing on board (this precludes interfaces which expect to control SCSI phases).
- d) Interpretation of sense data returned by SCSI devices shall be by the calling driver or program.
- e) Fully re-entrant code.  
NOTE: This is an obvious requirement for multitasking environments such as OS/2 or Unix but even in single tasking DOS applications, multithreaded I/O is required to achieve maximum performance. SCSI devices such as printers, communication ports and LAN interfaces are often serviced in the background under DOS. If an HBA cannot support multithreading, requests can be queued and serialized within the SIM module transparently to the XPT.
- f) Support of multiple HBAs.
- g) If optional features are not supported in a minimum functionality XPT and SIM, peripheral drivers shall be provided a means to determine what features are available.
- h) Providing an initialization service so that the process of identifying the attached SCSI devices need not be repeated by each peripheral driver which loads in the system.
- i) Providing a mechanism to abort I/O threads (at request of peripheral driver).
- j) Ability to issue multiple I/O requests from one or more peripheral drivers to a single Target/LUN.
- k) Providing peripheral drivers with a mechanism for allocating a Sense data area and for specifying the number of Sense bytes to be automatically requested on a CHECK CONDITION.

## 6. Transport

### 6.1 Accessing the XPT

The OS peripheral drivers access the XPT through a software call to a single entry point. The method for obtaining and using the entry point differs between operating systems.

The XPT is not involved in the reverse process to advise the peripheral driver of the completion of a request. The completion callback permits a direct return from the SIM to the peripheral driver (the exact method employed in callback is Operating System dependent).



The XPT is responsible to notify peripheral drivers of asynchronous events via the Asynchronous Callback mechanism

## 6.2 Initialization

The XPT is responsible for determining the interface configuration at power up initialization for the SIM drivers. Depending on the Operating System, the XPT may perform a scan of the attached SCSI peripherals automatically. See also the SCSI-2 X3.131-1990 Annex on Power Up Considerations.

The scan by the XPT/SIM would follow a pattern such as the following:

```

for all SCSI buses
  for all target IDs (excluding the initiator)
    find the device
    if device exists
      for all LUN's
        use Inquiry command and save returned information
      end for
    end if
  end for
end for

```

## 6.3 Callback on Completion

Callback on Completion refers to the XPT/SIM making a call to the routine addressed by the Callback on Completion pointer in the CCB. The callback is used by a peripheral driver in much the same manner as a hardware interrupt.

Callback routines have the same privileges and restrictions as hardware interrupt service routines.

The Callback on Completion routine is called to indicate that the Requested I/O is complete. The specific address of the CCB completed is passed to the callback routine.

## 6.4 SCSI Request Queues

Queues are used in systems where there is a need to manage many outstanding requests. There are various types of queues and each has different support needs.

A SCSI request queue can occur in the following places:

- o in the SIM
- o in the Target/LUN
- o in the peripheral driver

The SIM keeps a queue of all the CCB requests from the various peripheral drivers that access a LUN.

A SCSI device may be able to keep a large queue using Tag Queues, or a simple queue of one element.

A peripheral driver can also keep a queue e.g. a simple elevator sort, if the LUN does not support tagged queuing.

## 6.4.1 The Target/LUN and the Peripheral Driver

The peripheral driver is responsible for maintaining the queue(s) internal to the Target/LUN.

The SIM, acting on behalf of the peripheral driver, sends the appropriate commands or messages to manage the Target/LUN queue(s).

When the Target/LUN has completed an operation, the peripheral driver is advised by the SIM via a callback or by checking CAM status for completion.

The peripheral driver needs to be aware that there may be other peripheral drivers and other systems working with the same Target/LUN.

## 6.4.2 The SIM

The SIM maintains a queue for each LUN which is logically shared by all peripheral drivers. The queue may support tagged commands. Queue priority shall be supported.

## 6.4.3 SIM Queuing

### 6.4.3.1 SIM Queue Priority

When SIM Queue Priority=1, the SIM places the CCB at the head of the queue for the LUN, instead of at the end. One use of this CAM flag is during error handling. If the queue is frozen and a CCB with SIM Queue Priority=1 is received, the CCB shall be placed at the head of the queue and the queue remains frozen. When the SIM queue is released, any CCBs with SIM Queue Priority=1 are executed atomically, and in LIFO sequence.

To force step-by-step execution, the peripheral driver can set SIM Queue Freeze=1, so that when the queue is released and a CCB with SIM Queue Priority=1 is executed, the queue is re-frozen by the SIM at completion.

### 6.4.3.2 Tag Recognition

To support tagged queuing recognition the SIM maintains a reference between the CCB pointers and the Queue Tags for a LUN. By this means, the SIM can handle both the queue tag resource allocation and reconnection of the I\_T\_L\_Q nexus (see SCSI-2 X3.131-1990) for the CCB from a peripheral driver.

The peripheral driver is required to allow the SIM/XPT to handle the assignment of the queue tag ID for the request. The SIM assigns unique TAG IDs to the Target/LUN operation based on its internal reference table.

When a LUN that supports tagged queuing reconnects to the Initiator (SIM/HBA pair), it will send the SIMPLE QUEUE TAG message with the queue tag value for the I\_T\_L\_Q nexus. Using the returned queue tag ID, the SIM restores what is necessary to complete the SCSI transaction. The queue tag ID is freed by the SIM at the completion of the SCSI request.

### 6.4.3.3 Error conditions and Queues within the Subsystem

The SIM shall place its internal queue for a LUN into the frozen state for any status other than Request Completed Without Error and Request in Progress. When a LUN's queue is in the frozen state, the SIM shall not dispatch any CCBs

to that LUN. Peripheral drivers can still send CCBs to the SIM for the LUN, or any other LUN. Any new CCBs received by the SIM shall be placed at the end of the queue, unless SIM Queue Priority=1 forces them to the head,

Following a Check Condition or Command Terminated status, the target's LUN queue is also frozen, and all other tagged commands stay in the queue until the allegiance condition is cleared. The allegiance condition is either cleared by an incoming command or following the return of sense data for the same initiator.

Since the SIM is the initiator, the SIM's internal queue shall go into a frozen state so that the pending sense information in the LUN will not be discarded. The SIM holds it's internal LUN queue in the frozen state until a Release SIM Queue CCB is received.

Using the Callback on Completion pointer contained in the CCB the SIM returns control of the CCB to the peripheral driver along with CAM Status indicating the frozen queue condition and other information.

The peripheral driver acts upon the information returned via the CCB. In the event that there is not a valid pointer in the callback field, the peripheral driver that originated the CCB shall retain responsibility for the CCB by watching the CAM Status field. The setting of the Autosense bit in the CAM flags does not affect how the SIM handles freezing the SIM's internal queue i.e. the Request Sense command issued by the SIM to recover status for Autosense does not release the SIM queue.

If the peripheral driver has to perform recovery with the LUN, a CCB can be placed at the head of the queue by setting SIM Queue Priority=1, and the SIM queue released. If the peripheral driver has other pending CCBs in the queue which it does not want to be sent to the LUN (depending on the cause of the Check Condition), then it can use a CAM Flag to freeze the queue upon completion of the CCB at the head of the queue. A SIM may reject a CCB with SIM Queue Freeze=1 if the queue is not frozen at the time the CCB is received.

## 6.5 SIM Handling of SCSI Resets

The CAM shall not define support for the "Soft Reset" SCSI option, but implementors may use the services of the SIM to provide vendor-specific support.

Following a SCSI Bus Reset, the SIM shall:

- a) Block Path IDs to the reset bus i.e. new CCBs are rejected with status of CAM Busy.
- b) Return all outstanding CCBs with status of SCSI Reset.
- c) Unblock all Path IDs for the bus.
- d) Call: `xpt_async(opcode=reset, path_id=bus that was reset, target_id=-1, lun=-1, buffer_ptr=null, data_cnt=0)`
- e) Resume normal processing of CCBs.

## 6.6 Asynchronous Callback

In an event such as a SCSI Bus Reset or an Asynchronous Event Notification the XPT has to be able to make a callback to the peripheral driver(s), even though there may be no CCBs active for the peripheral driver(s).

Callback routines have the same privileges and restrictions as hardware interrupt service routines.

During system startup and driver initialization, the peripheral driver should register an Asynchronous Callback routine for all the SCSI devices with which it is working. In order for a peripheral driver to receive asynchronous callbacks, it shall issue a **Set Asynchronous Callback CCB with the Asynchronous Event fields set to 1** for those events the peripheral driver wishes to be notified of through an asynchronous callback. **The peripheral driver is required to explicitly register for the path IDs, targets, and LUNs. The use of a wildcard is not supported for the Set Asynchronous Callback CCB.**

It is required that the Asynchronous Callback field be filled in with the callback routine address if any of the Asynchronous Events Enabled bits are set. The peripheral driver can de-register its Asynchronous Callback for a particular SCSI device by issuing the Set Asynchronous Callback CCB with the Events field cleared to zero and the Callback pointer containing the Callback Routine address of the peripheral driver issuing the request. All XPTs must provide the capability for any SIM to support asynchronous callback, but a given SIM does not have to support each (or any) of the Asynchronous Events Enabled bits.

Upon detection of a supported enabled event, the SIM shall do the following once for each detected event:

- a) Classify the event: determine the opcode which is the same as the encoded bit number of the Asynchronous Events Enabled.
- b) Format the associated data within an internal, to the SIM, local buffer, e.g. the sense data received from an AEN.  
NOTE: This is a multiple processor "lock" point.
- c) Perform the XPT reverse routing required by the event. The SIM will call the Async Callback entry point in the XPT:

```
long xpt_async(opcode, path_id, target_id, lun, buffer_ptr, data_cnt)
```

All of the arguments, other than the pointer, are long values of 32 bits. The value of -1 in Path, Target and LUN can be used as a wild card. A null buffer pointer value and a count of 0 are valid for opcodes that do not require any data transfer.

NOTE: This call to the XPT is a multiple processor "lock" point.

Using the Path ID, Target, and LUN information from the `xpt_async()` call, the XPT scans its internal tables looking for "matches" with what the peripheral drivers had registered for using the Set Async Callback CCB (see 8.2.4). When a match is found, either exactly or with a wild card of "-1," the XPT shall copy the data for the opcode, if available, into the area reserved by the peripheral driver and then call the peripheral driver's Async Callback routine.

The arguments to the peripheral driver's Async Callback routine are the same

as the xpt\_async() routine though:

- the buffer\_ptr value shall be the peripheral driver's buffer
- the data\_cnt shall either be what the XPT had to transfer from the SIM's buffer or the limit of the peripheral driver's buffer.

Almost all of the information relating to the different opcodes can be included in the Path ID, Target and LUN arguments. The only opcodes that require an additional buffer area are AEN, Load SIM and Unload SIM. Table 6-1 lists the opcodes and the expected data requirements for the number of bytes to be transferred.

TABLE 6-1 ASYNC CALLBACK OPCODE DATA REQUIREMENTS

Event	Opcode	Path ID	Target	LUN	Data Cnt
Unsol. SCSI Bus Reset	0x0001	Valid	n/a	n/a	n/a
Unsol. Reselection	0x0002	Valid	Valid	Valid	n/a
reserved	0x0004				
SCSI AEN	0x0008	Valid	Valid	Valid	Min. 22
Sent BDR to Target	0x0010	Valid	Valid	n/a	n/a
SIM Module Loaded	0x0020	XPT ID	n/a	n/a	Min. 1
SIM Module Unloaded	0x0040	XPT ID	n/a	n/a	Min. 1
New Devices Found	0x0080	Valid	n/a	n/a	n/a

The AEN data requirements are a minimum of 22 bytes of buffer space. This space includes the 4 bytes required by the AEN Data Format and 18 bytes defined by the Sense Data Format (see SCSI-2 X3.131-1990).

The Load SIM and Unload SIM data requirements are a minimum of 1 byte. This byte contains the Path ID for the SIM. This Path ID is different than the path id argument. The argument contains the unique XPT ID of 0xFF. The XPT ID is the ID used by the peripheral driver to register for async notification.

If there is valid data placed in the generic data buffer by the XPT/SIM, the peripheral driver is required to save or discard that data before returning control to the XPT/SIM.

### 6.7 Autosense

Autosense causes sense data to be retrieved automatically if a Check Condition is reported in the SCSI Status field. On a Check Condition, a SCSI Request Sense command is constructed and sent to the same target. The location and amount of the Sense data is specified in the Sense Info Buffer Pointer and Length fields respectively of the SCSI I/O Request CCB. If the length field is 0 or the buffer field is Null, the Request Sense command shall still be issued, but with a data allocation length of 0 (this should only be done by the peripheral driver when it is not interested in the sense information).

After completing the Request Sense sequence the CAM Status and SCSI Status fields contain the status of the original command (which caused the Check Condition).

The target can return fewer than the number of Sense bytes requested. This is not reported as an error, and Sense Status shall be flagged as valid.

### 6.8 Loadable Modules

Some operating system environments provide the ability to load or unload software drivers, thus peripheral drivers or SIM modules can be loaded dynamically. In such systems, the XPT module (typically supplied by the OS vendor) is either part of the system or must be loaded first.

The XPT, as part of a loadable OS, exports its "label," which is to used as a reference by the other loadable modules. The XPT manages the loading of SIMs and provides the common access point for peripheral drivers to register a loaded or unloaded SIM.

When a peripheral driver is loaded, it can go through its initialization process (see OSD initialization), call the XPT initialization point and then query the XPT for the HBAs that are present in the system and targets that have been identified as being on the SCSI channels.

When a SIM is loaded, the SIM and XPT have to work together to get the SIM entered into the internal tables and have the SIM initialized.

The SIM shall call the XPT once for each supported bus in order to obtain the Path ID for that bus.

```
long xpt_bus_register(CAM_SIM_ENTRY *)
```

The argument is the pointer for the data structure defining the entry points for the SIM. The value returned is the assigned Path ID; a value of -1 indicates that registration was not successful.

The SIM shall call the XPT once to de-register the bus for a given Path ID:

```
long xpt_bus_deregister(path_id)
```

The argument is the Path ID for the bus being de-registered. A return value of zero indicates the bus is no longer registered, any other value indicates the call was unsuccessful.

When the XPT is called it will update its internal tables and then call the sim\_init(path\_id) function pointed to by the CAM\_SIM\_ENTRY structure. The initialization for the loaded SIM is no different than for a SIM statically included in the kernel at boot time. After the SIM has gone through the initialization process the XPT shall scan the SCSI bus in order to update its internal tables containing Inquiry information.

Peripheral drivers can request to be informed when a SIM is registered or de-registered via the Async Callback feature (see 6.6 and 8.2.4).

The CAM\_SIM\_ENTRY table is used to define the entry points for the SIMs.

```
typedef struct
{
    long (*sim_init)(); /* pointer to the SIM init routine */
    long (*sim_action)(); /* pointer to the SIM CCB go routine */
} CAM_SIM_ENTRY;
```

## 7. OSD (Operating System Dependent) Operation

### 7.1 UNIX Operating System

The CAM subsystem is intended to provide a set of services for third-party vendors.

There are several sets of modules for Unix:

- peripheral drivers that are device class specific
- a configuration driver for initialization
- the XPT
- SIMs that are HBA-specific

Each member of these sets is treated as a UNIX driver and is linked into the kernel. The XPT and configuration driver (which is responsible for initialization) are OS-vendor specific; other drivers may come from any source.

At kernel configuration and link time the `cam_conftbl[]` is created and contains entry points for the SIMs, which are used by the XPT.

The `cam_conftbl[]` is used by the XPT/configuration driver to call routines and pass CAM parameters between them e.g. the Path ID contained in the CCB created by the peripheral driver is used to index into the `cam_conftbl[]`. The entry point for the selected SIM, `sim_action()` is called with a pointer to the CCB as an argument.

The `cam_edt[]` data structure is used and created during the initialization process to contain the necessary information of all the targets found on all the HBAs during the init sequence.

The CAM Flags used are as described in Table 9-2.

#### 7.1.1 Initialization

The initialization of the XPT and SIMs is under the control of the configuration\_driver.

Due to the different Unix-based systems (BSD and System V), there is no common initialization process that can control the order of calls to the peripheral driver's and configuration driver's `init()` routines. It is necessary to make sure that the subsystem is initialized before any requests can be serviced from the peripheral drivers. Due to this constraint when the peripheral driver's initialization routines are called the driver shall call the `xpt_init()` routine. If the subsystem is not yet initialized, the XPT shall call the configuration driver to formally initialize the subsystem. Once the subsystem is set up, either from a previous `xpt_init` call or the configuration\_driver being called, all subsequent `xpt_init` calls shall simply return.

When the configuration driver is called for initialization, it uses the `cam_conftbl[]` entry structures. The configuration driver makes the `init()` routine calls, to the XPT, and to each SIM in turn, allowing them to initialize. The initialization routine for the SIM is called with its Path ID as the argument. Interrupts shall be disabled or blocked by the configuration\_driver during the initialization process.

After the initialization process has been completed, the configuration driver obtains information about each SIM, HBA, and target device detected, and maintains a table, the `cam_edt[]`, of these devices. The information is obtained by using CCBs through the CAM interface.

Once the CAM subsystem is initialized and the `cam_edt[]` set, the peripheral drivers can use the subsystem. This allows them to determine what devices are known and make appropriate memory allocations and resource requests of the XPT.

The SCSI-2 Inquiry command shall be issued to all Target/LUNs on the attached interfaces, and shall contain an allocation length of 36 bytes, which is sufficient to transfer the device information and the product information. The EVPD and Page code fields in the Inquiry command shall be set to 0. It is assumed that the responding devices will return the Inquiry data, even though the device may not be ready for other commands. A limited number of retries will be done for devices that return Busy Status following the Inquiry command. If the retry limit is reached, the status of the device in the XPT will be set to "Not Found". The Inquiry command shall be the only command issued by the XPT to the devices during initialization.

#### 7.1.2 Accessing the XPT

##### 7.1.2.1 From the Peripheral Driver

The XPT provides functions to obtain CAM system resources for the peripheral driver. These functions are used to allocate and free CCB resources and to allocate and free DMA resources.

There are two routines used in the handling the CCB resources. The two routines are:

```
CCB *xpt_ccb_alloc() and
void xpt_ccb_free(CCB *):
```

- The `xpt_ccb_alloc()` routine returns a pointer to the allocated CCB. The peripheral\_driver can now use this CCB for it's SCSI/XPT requests.
- The `xpt_ccb_free()` routine takes a pointer to the CCB that the peripheral\_driver has finished with, and can now be returned to the CAM subsystem CCB pool.
- The pointer to the CCB returned from the `xpt_ccb_alloc()` call shall be large enough to contain any of the possible XPT/SIM function request CCBs.
- **The CCB can only be used i.e. sent to the XPT, once. Once the CCB has completed it shall be returned using the `xpt_ccb_free()` routine.**

All returned status information is obtained at the callback point via the CAM and SCSI status fields.

##### 7.1.2.2 From the SIM

The SIMs obtain requests from the XPT as they are passed across from the peripheral driver, via a routine included in the SIM's configuration information. The field in the configuration table is declared as "void (\*sim\_action)(CCB \*)." The XPT does not modify CCBs or CDBs. The XPT shall intercept those CCBs which must be redirected to the configuration\_driver (Get Device Type, and Set Device Type).

### 7.1.3 Callback on Completion

The Callback on Completion field in the CCB is a structure that is platform specific, but always contains at least a callback function pointer, named `cbfcn`, and declared as "void (\*cbfcn)(CCB \*)." The argument to `cbfcn` shall be the address to the CCB.

The Disable Callback on Completion feature is not supported.

### 7.1.4 Pointer Definition in the UNIX Environment

Pointers in the CAM environment are treated as any other pointer in a given UNIX implementation. For the 80386 platforms, pointers are 32-bit virtual addresses into a flat address space.

### 7.1.5 Request Mapping Information

This field is expected to contain a pointer to the `buf` structure that the SCSI I/O CCB was created for. This copy of the `buf` structure pointer, `bp`, is used by the SIM to get to the I/O mapping information needed to access the data buffers allocated by the application program. A value of NULL is allowed if there is no need for the SIM to map the data buffer addresses i.e. data count is zero, the buffer is internal to the kernel, or the addresses are physical.

### 7.1.6 XPT Interface

The XPT interface provides functions that peripheral drivers and SIM modules can access in order to transfer information and process user requests. The following defines the entry points, and describes the required arguments and return values.

#### 7.1.6.1 Functions for Peripheral Driver Support

a) `long xpt_init()`

This routine is called by the peripheral driver to request that the XPT and sub-layers be initialized. Once the sub-layers are initialized any subsequent calls by other peripheral drivers shall quickly return.

There are no arguments and the return code is either Success or Failure.

b) `CCB *xpt_ccb_alloc()`

This routine is used whenever a peripheral driver needs a CCB (the common data structure for processing SCSI requests). It returns a pointer to the allocated CCB which the peripheral driver can now use as the CCB for its SCSI/XPT requests. The returned CCB shall be properly initialized for use as a SCSI I/O Request CCB. The SIM Private Data area shall have been already set up to be used by the XPT and SIM, and shall not be modified by the peripheral driver. It is recommended that the CCB be returned to the XPT following its use, and that CCBs not be re-used.

There are no arguments and the return value is a pointer to an initialized CCB.

c) `void xpt_ccb_free(CCB *)`

This routine takes a pointer to the CCB that the peripheral driver has finished with so it can be returned to the CAM subsystem CCB pool.

The argument is the pointer to the CCB to be freed, there is no return code.

d) `long xpt_action(CCB *)`

All CAM/SCSI CCB requests to the XPT/SIM are placed through this function call. All returned CAM status information is obtained at the callback point via the CAM and SCSI status fields.

The argument is a pointer to the CCB, and the return code is either Success or Failure.

#### 7.1.6.2 Functions for SIM Module Support

a) See 6.8 for loadable module support:

`long xpt_bus_register(CAM_SIM_ENTRY *)`

`long xpt_bus_deregister(path_id)`

b) `long xpt_async(opcode, path_id, target_id, lun, buffer_ptr, data_cnt)`

The SIM calls this routine to inform the XPT that an async event has occurred and that there may be peripheral drivers which need to be informed.

- The `opcode`, `path_id`, `target_id`, `lun`, and `data_cnt` arguments are long 32-bit values.
- The `path_id`, `target_id`, and `lun` define a nexus for the Async Callback.
- The `opcode` contains the value for what has happened.
- The `buffer_ptr` and `data_cnt` are used to inform the XPT where and how much data is associated with the `opcode`.

The return code is either Success or Failure.

### 7.1.7 SIM Interface

The SIM interface provides functions to the XPT, and should never be accessed directly by the peripheral driver. Each vendor's SIM should provide a publicly-defined entry structure such as `CAM_SIM_ENTRY cse_vendorname`.

The following defines the entry points, and describes the required arguments and return values.

a) `long sim_init(pathid)`

This routine is called by the XPT to request that the SIM be initialized. There are no arguments and the return code is either Success or Failure.

b) `long sim_action(CCB *)`

All CCB requests to the SIM are placed through this function call. All returned CAM status information is obtained at the callback point via the CAM and SCSI status fields.

The argument is a pointer to the CCB, and the return code is either Success or Failure.

## 7.2 Novell Operating System

Novell NetWare 386 drivers are called NLMs (NetWare Loadable Modules). These modules are registered and linked dynamically with NetWare 386: they are registered after the server is running and may be **unloaded** at any time.

The NetWare 386 CAM subsystem consists of 3 sets of NLMs:

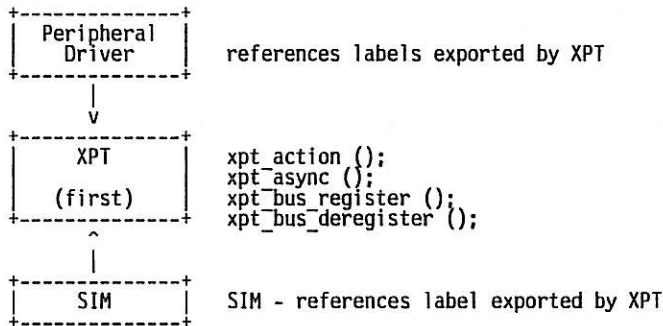
- peripheral drivers (NLMs) that are device class specific
- the XPT router and SIM maintenance NLM
- SIM NLMs that are HBA-specific

The peripheral drivers and SIMs communicate with the XPT through labels exported by the XPT when it is registered.

The CAM Flags used are as described in Table 9-2.

### 7.2.1 Initialization

As the Novell dynamic linker will not allow an NLM to register if it makes references to a label it cannot resolve, the order in which the NLMs register is important. The XPT module exports **four** entry points when it is registered, and both peripheral drivers and SIM modules make references to them. The XPT shall be registered first, after which either peripheral drivers or SIMs may be registered.



For an overview of SIM registration with the XPT see 6.8. For an overview of peripheral driver registration with the XPT see 6.6 and 8.2.4.

When **NetWare 386** loads a **SIM**, it shall call the initialization routine **specified in the Novell linker definition file**. At this point the SIM can perform its initialization functions.

As part of initialization the SIM shall call the `xpt_bus_register` function once for each HBA it will support, to register the address of its entry point with the XPT and to get a path ID for each HBA from the XPT. The XPT then adds

this SIM to its internal tables so it can route requests to the new SIM. The XPT also notifies all peripheral drivers that registered an asynchronous callback routine with the XPT (with the SIM Module Registered bit set), that a new path ID exists. Upon receiving this message the peripheral drivers can check for new devices on this path.

When **NetWare 386** loads a peripheral driver, the initialization routine **specified in the linker definition file shall be called**. At this time, the driver needs to determine which, if any, SIMs are registered.

The peripheral driver sends a Path Inquiry CCB to each path to determine if a SIM is registered. If a valid response is returned the peripheral driver checks for devices that it will support on that path. If the peripheral driver supports any devices on this path, it shall register an asynchronous callback routine and specify the SIM registration in the opcode field so that if the SIM is de-registered, the peripheral driver shall be notified. In addition, a peripheral driver should also register for SIM registration to alert the driver of the need to locate devices on a newly added SIM module.

### 7.2.2 De-Registration

When a SIM de-registers, it shall call the `xpt_bus_deregister()` function once for each path the SIM supports. The XPT then calls every peripheral driver that has registered an asynchronous callback routine with the SIM Module De-Registered bit set on this path. Peripheral drivers then notify NetWare 386 that the drives on this path are in an inactive state. The XPT will then remove the path from its internal tables and block further peripheral driver requests on this path.

If a peripheral driver de-registers, it needs to notify the XPT module so that the dependency tables can be updated. This is done by registering an asynchronous callback routine with the opcode set to zero. The XPT will then remove this driver from its callback tables.

The XPT can only be **unloaded** after all peripheral drivers and SIM modules have been de-registered. NetWare 386 will not allow an NLM to **unload** if it has exported labels that other NLMs are using. As all SIM and peripheral drivers refer to labels exported by the XPT, NetWare 386 will not allow the XPT to **unload** until all the SIMs and peripheral drivers have been **unloaded**, at which point there is nothing left for the XPT to support and it can be safely de-registered.

### 7.2.3 Accessing the XPT

NetWare 386 allows an NLM to export functions which NLMs registered at a later time can reference. An NLM calls an exported function in the same way it calls any other function. The **C language calling convention is assumed**. In order for communication between the peripheral drivers, XPT, and SIM modules to work correctly the names of the XPT entry points have to be constant.

The entry points in the XPT module are:

- `xpt_action ()` accepts CAM blocks from the peripheral driver and routes them to the correct SIM
- `xpt_async ()` is used by the SIM module to notify the XPT when an asynchronous event occurs.
- `xpt_bus_register ()` is used to register the SIM with the XPT and obtain a Path ID.

- `xpt_bus_deregister ()` is used to **unload** the SIM associated with the Path ID.

#### 7.2.4 Hardware Registration

The SIM module needs to do the actual registration of the host adapter with NetWare. Since only one SIM may support a given host adapter this prevents any hardware options from being registered twice. The SIM does not register any devices with NetWare, only the hardware options used by the card e.g. interrupt line, base address, DMA etc.

Interrupts generated by the host adapter will be handled by the SIM module, so the SIM must also register its interrupt service routine with NetWare.

A peripheral driver registers a logical card with NetWare 386 for each path id it supports. This logical card uses no hardware resources, but does have entry points IO and IOCTL requests from NetWare. The peripheral driver also reports the devices that it will support to NetWare.

The XPT does not register any hardware or devices with NetWare 386. It loads as a driver, but does not register any IOPOLL or IOCTL entry points.

#### 7.2.5 Miscellaneous

It is the responsibility of the peripheral driver to allocate memory for its CCB blocks. Normally the peripheral driver needs to keep one CCB structure for each device it will support, so the memory can be allocated in the DiskStructure provided by NetWare 386 when a device is added to the system.

Since fast disk channels are essential for a NetWare 386 server, peripheral drivers should never poll the CAM status field to wait for completion. The driver should send the CCB to the XPT module and then either do more work, or exit immediately. The SIM module will call the function whose address is in the callback field of the CCB block when the request is finished. The callback function runs at interrupt level, so it cannot call any NetWare 386 routines that are "blocking" or the file server will abend. See the Novell Disk Driver manual for details on blocking and non-blocking levels.

#### 7.3 DOS (Disk Operating System)

Under DOS, a software interrupt is used to access any of the XPT or SIM functions, which are combined into a single module.

The routing functions of the XPT are performed by the DOS concept of "interrupt vector chaining." During execution, an XPT/SIM module determines if a particular CCB is one that it should handle. If not, it routes the CCB to the previous "owner" of the interrupt vector.

The CAM flags used by the DOS XPT/SIM are described in Table 9-2.

##### 7.3.1 Initialization

During initialization, the XPT/SIM modules should be loaded as character device drivers.

As character device drivers are required by DOS to have unique names, the 8-character device name should be "\$\$CAMxxx", where xxx is the ASCII decimal

numeric value of the lowest path ID supported by this XPT/SIM module.

The programming examples in this clause are used to assist the reader's understanding. Implementations do not need to use the same code, but they are required to accomplish the same goals.

##### 7.3.1.1 Multiple XPTs

The pseudocode for the XPT initialization sequence is as follows:

```

Get INT 4Fh interrupt vector;
Save this address for chaining;
IF there is a CAM XPT already installed (see 7.3.2.1)
    Perform PATH INQUIRY (Path ID=0FFh) to get Highest Path ID;
    First Path ID = Highest Path ID + 1;
ELSE
    First Path ID = 0;
END IF;
Count number of Path IDs needed;
IF no HBAs to support (Count = 0)
    Exit initialization without installing driver;
END IF;
Set INT 4Fh interrupt vector to point to CAM entry point;
Save Highest Path ID used (First Path ID + Count - 1);
Set character device name to "$$CAMxxx",
    where xxx=First Path ID;
Perform all necessary HBA initialization;
FOR each SCSI Bus supported;
    FOR each SCSI ID (excluding initiator)
        IF device exists
            FOR each LUN
                Perform INQUIRY to get PDT for table;
            END FOR;
        END IF;
    END FOR;
END FOR;
    
```

##### 7.3.1.2 Device Table Handling

The XPT/SIM is only required to keep the peripheral device type of the devices connected to the supported SCSI bus(es).

##### 7.3.2 Accessing the XPT

There are various mechanisms used to access XPT or SIM functions from peripheral drivers or application programs.

##### 7.3.2.1 Testing for the presence of the XPT/SIM

Peripheral drivers and applications can check for the presence of an XPT/SIM module by performing a "check install" function such as:

```

On entry:
AX = 8200h
CX = 8765h
DX = CBA9h
    
```

## On return:

AH = 0 (if successful)  
 CX = 9ABCh  
 DX = 5678h  
 ES:DI = address of character string "SCSI\_CAM"  
 All other registers unaffected.

The following routine checks for the presence of an XPT/SIM module. It returns a value of 1 if a module is found and a value of 0 if not found.

```

CHK_FOR_CAM PROC NEAR
MOV CX,8765H ; load l.s.w. of signature
MOV DX,0CBA9H ; load m.s.w. of signature
MOV AX,8200H ; load "check install" code
INT 4FH ; perform "check install"
CMP AH,0 ; function supported?
JNE NOT_THERE ; if not, no xpt/sim
CMP DX,5678H ; check m.s.w. of signature
JNE NOT_THERE ; if invalid, no xpt/sim
CMP CX,9ABCH ; check l.s.w. of signature
JNE NOT_THERE ; if invalid, no xpt/sim
CLD ; set direction flag
MOV CX,8 ; load string length
MOV SI,OFFSET SCSI_CAM ; get string address
REPE CMPSB ; compare strings
JNE NOT_THERE ; if strings differ, no xpt/sim
MOV AX,I ; load "found" status
RET ; return to caller
NOT_THERE: MOV AX,0 ; load "not found" status
RET ; return to caller
CHK_FOR_CAM ENDP
SCSI_CAM DB 'SCSI_CAM' ; string to find
  
```

## 7.3.2.2 Sending a CCB to the XPT

Once it is determined that an XPT/SIM module is present, the peripheral driver or application can access the XPT/SIM functions by sending a CCB to the XPT/SIM:

## On entry:

ES:BX = address of the CCB  
 AX = 8100h

## On return:

AH = 0 if successful  
 = 1 if invalid CCB address (segment-offset=0)  
 All other registers unaffected.

NOTE: The SIM may complete and return control to the location pointed to by the Callback on Completion field in the CCB before the software interrupt returns.

The following routine sends a CCB to the XPT/SIM module. It returns a value of 0 if successful and 1 if not.

```

SEND_CCB PROC NEAR
MOV AX,8100H ; load "send ccb" function
MOV ES,SEGMENT CCB ; load segment of ccb
MOV BX,OFFSET CCB ; load offset of ccb
INT 4FH ; call xpt/sim module
SHR AX,8 ; put return code in al
RET ; return to caller
SEND_CCB ENDP
  
```

## 7.3.3 Callback on Completion

When an I/O operation has completed, a XPT/SIM module shall make a FAR call to the routine which had its address passed in the Callback on Completion field of the CCB. The first 4 bytes of this field are used to indicate the routine's address in the Intel Segment:Offset format. When the callback is made, hardware interrupts shall be disabled and ES:BX shall point to the completed CCB.

## 7.3.4 Asynchronous Callbacks

There are some differences in the DOS XPT/SIM implementation of Asynchronous Callbacks as compared with the description in 6.6.

The DOS XPT/SIM does not support the SIM Module Loaded and SIM Module Unloaded opcodes reported by the XPT/SIM module when the Asynchronous Callback Routine is called.

The Set Async Callback CCB is held by the XPT/SIM until it is "de-registered." This is accomplished by sending another Set Async Callback CCB to the XPT/SIM with all of the Asynchronous Event Enables reset and the address of the original Set Async Callback CCB in the Peripheral Driver Buffer Pointer field. At that point the original CCB shall be dequeued and both CCBs shall be returned to the peripheral driver or application.

NOTE: There is an implication here that a peripheral driver or application which wishes to be notified when the specified asynchronous event occurs, has to register separately with each path ID.

The Peripheral Driver Buffer Pointer and Size of Allocated Peripheral Buffer fields in the Set Async Callback CCB are considered as Private Data by the XPT/SIM, to be used for CCB queuing.

When an Asynchronous event occurs that is enabled by the bits in the Asynchronous Event Enables field of the Set Async Callback CCB, the virtual address specified by the Asynchronous Callback Pointer field shall be called with the following registers:

## On entry:

AH = opcode as specified in Table 6-1.  
 AL = path ID that generated the callback.  
 DH = target ID that caused event (if applicable).  
 DL = LUN that caused event (if applicable).  
 CX = data byte count (if applicable).  
 ES:BX = address of data buffer (if applicable).

## On return:

All registers shall be preserved.



It is the responsibility of the peripheral driver or application to copy any or all required data out of the data buffer into a local buffer before returning from the Asynchronous Callback routine.

7.3.5 Pointer Definition

All pointers shall be passed to the XPT/SIM as segment:offset type virtual addresses.

7.4 OS/2 (Operating System 2)

Microsoft has documented LADDR as a generic I/O interface which supports many device interfaces, not only SCSI. The control blocks and their method of operation are defined in OS/2 Technical Reference Manuals which are available from Microsoft.

The OS/2 equivalent to the SIM is a BID (Bus Interface Driver).

The OS/2 equivalent to the CCB is an SRB (SCSI Request Block).

The CCB and the SRB share many common fields. The fields in the CCB are designated as OSD if they vary between OS/2 and other operating systems.

For further information on how peripheral drivers use the CCB/SRB and other SIM/BID capabilities of OS/2, it is necessary to use information available from Microsoft.

The CAM Flags used are as described by LADDR documentation.

8. CAM Control Blocks

The CCBs used by drivers and applications to request functions of the XPT and SIM have a common header, as shown in Table 8-1.

TABLE 8-1 CAM CONTROL BLOCK HEADER

Size	Dir	
4	0	OSD Address of this CCB
2	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		Common
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)

The sequence of the fields in the data structures will be consistent between vendors, but not necessarily the binary contents. The size and definition of the fields in the data structures can vary between operating systems and hardware platforms, but the vendors are expected to provide compiler

definitions which can be used by third-party attachments.

Several fields in the CCB are pointers, and their meaning is dependent on the OS which is being supported. In general, these pointers are interpreted as either virtual or physical addresses.

Additional bytes beyond the CCB Header are dependent on the Function Code.

Most SCSI messages are handled transparently by the SIM, but in some cases, the peripheral driver has been given the ability to force the SIM to issue a message. Table 8-2 summarizes the message support.

TABLE 8-2 SUPPORT OF SCSI MESSAGES

Abort	Discretely supported by function codes
Abort Tag	Discretely supported by function codes
Bus Device Reset	Discretely supported by function codes
Clear Queue	Not Supported
Command Complete	Transparently supported by SIM
Disconnect	Transparently supported by SIM *
Identify	Transparently supported by SIM
Ignore Wide Residue	Transparently supported by SIM
Initiate Recovery	Not Supported
Initiator Detected Error	Transparently supported by SIM
Linked Command Complete	Transparently supported by SIM
Message Parity Error	Transparently supported by SIM
Message Reject	Transparently supported by SIM
Modify Data Pointer	Transparently supported by SIM
No Operation	Transparently supported by SIM
Queue Tag Messages	
Head of Queue Tag	Discretely supported by function codes
Ordered Queue Tag	Discretely supported by function codes
Simple Queue Tag	Discretely supported by function codes
Release Recovery	Not Supported
Restore Pointers	Transparently supported by SIM
Save Data Pointers	Transparently supported by SIM
Synch Data Transfer Request	Transparently supported by SIM *
Terminate I/O Process	Discretely supported by function codes
Wide Data Transfer Request	Transparently supported by SIM

\* Issuing this message influenced by peripheral driver via CAM flags

8.1 CCB Header

The Function Codes used to identify the XPT service being requested are listed in Table 8-3.

8.1.1 CAM Control Block Length

See 9.1.1.

8.1.2 XPT Function Code

TABLE 8-3 XPT FUNCTION CODES

Code	
00-0F	Common Functions
00h	NOP
01h	Execute SCSI I/O (see 9.x)
02h	Get Device Type
03h	Path Inquiry
04h	Release SIM Queue
05h	Set Async Callback
06h	Set Device Type
07-0F	reserved
10-1F	SCSI Control Functions
10h	Abort SCSI command
11h	Reset SCSI Bus
12h	Reset SCSI Device
13h	Terminate I/O Process
14-1F	reserved
20h	Engine Inquiry (see 11.x)
21h	Execute Engine Request
22-2F	reserved
30-3F	Target Mode (see 10.x)
30h	Enable LUN
31h	Execute Target I/O
32-3F	reserved
40-7F	reserved
80-FF	Vendor Unique

If a Function Code which is not supported is issued to the XPT, the XPT shall complete the request and post CAM Status of Invalid Request.

8.1.3 CAM Status

See 9.1.3.

8.1.4 Path ID

See 9.1.4.

8.1.5 CAM Flags

The CAM Flags qualify the Function to be executed, and vary by Function Code. See 9.1.5.

8.2 Function Codes

8.2.1 Get Device Type

This function is executed at driver initialization in order to identify the targets they are intended to support e.g. A CD ROM driver can scan each Target/LUN address on each installed HBA to look for the CD ROM device type.

TABLE 8-4 GET DEVICE TYPE CCB

Size	Dir	Get Device Type
4	0	OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
4	0	<b>Inquiry Data Pointer</b>
1	I	Peripheral Device Type of Target/LUN

The information on attached SCSI devices is gathered at power on by the XPT (to eliminate the need for each driver to duplicate the effort of scanning the SCSI bus for devices).

The Peripheral Device Type is a 1-byte representation of Byte 0 of SCSI Inquiry Data i.e. bits 7-5=000.

If the Inquiry Data Pointer contains a value other than Null, it is a pointer to a buffer in the peripheral driver's data space large enough to hold the 36 bytes of Inquiry data associated with the Target/LUN. The data shall be copied from the internal tables of the XPT to the peripheral driver's buffer.

This function shall return non-zero CAM Status.

- CAM Status of Request Completed Without Error indicates that the specified device is installed and the **Peripheral Device Type** field is valid.
- CAM Status of SCSI Device Not Installed indicates that the **Peripheral Device Type** field is not valid.
- CAM Status of Invalid Path ID indicates that the Path ID is invalid.

Drivers are always able to use SCSI I/O requests to check for devices which may not have been found at power up.

8.2.2 Path Inquiry

This function is used to get information on the installed HBA hardware, including number of HBAs installed. To obtain further information on any other HBAs attached, this function can be issued for each HBA.

If the Path ID field of the CCB has a value of FFh on a PATH INQUIRY request, then the only field that shall be valid upon return to the caller is the Highest Path ID Assigned field. In addition, this field shall not be valid if the Path ID field in the CCB contains a value other than FFh.

TABLE 8-5 PATH INQUIRY CCB - Part 1 of 2

Size	Dir	Path Inquiry
4	0	- - - - - OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		- - - - - Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
1	I	Features Supported
		Version Number
		00-07h Prior to Rev 1.7
		08h Implementation Version 1.7
		09-FFh Rev No e.g. 23h = 2.3
1		SCSI Capabilities
		7 Modify Data Pointers
		6 Wide Bus 32
		5 Wide Bus 16
		4 Synchronous Transfers
		3 Linked Commands
		2 reserved
		1 Tagged Queuing
		0 Soft Reset
1		Target Mode Support
		7 Processor Mode
		6 Phase Cognizant Mode
		5-0 reserved
1		Miscellaneous
		7 0=Scanned Low to High
		1=Scanned High to Low
		6 0=Removables included in scan
		1=Removables not included
		5 1=Inquiry data not kept by XPT
		4-0 reserved

66

TABLE 8-5 PATH INQUIRY CCB - Part 2 of 2

Size	Dir	Path Inquiry
2	I	HBA capabilities
14	I	Engine count
4	I	Vendor Unique
4	I	Size of Private Data Area
		Asynchronous Event capabilities
		31-24 Vendor Unique
		23- 8 reserved
		7 New Devices found during rescan
		6 SIM module De-Registered
		5 SIM module Registered
		4 Sent Bus Device Reset to Target
		3 SCSI AEN
		2 reserved
		1 Unsolicited Reselection
		0 Unsolicited SCSI Bus Reset
1	I	Highest Path ID Assigned
1	I	SCSI Device ID (of Initiator)
1		reserved
1		reserved
16	I	Vendor ID of SIM-supplier
16	I	Vendor ID of HBA-supplier
4	0	OSD Usage

In some Operating System environments it may be possible to dynamically load and unload SIMs, so Path IDs may not be consecutive from 0 to the Highest Path ID Assigned.

The Path ID value of FFh is assigned as the address of the XPT.

The SCSI Capabilities field is a duplicate of the Byte 7 field in Inquiry Data Format.

The OSD Usage Pointer field is provided for OS-specific or platform-specific functions to be executed by the SIM. The contents of this field are vendor-specific and are not defined by this standard.

In some environments, the Private Data value returned may be zero because the OSD has central allocation of private data requirements, or it is a fixed size as defined by the OSD vendor.

See the vendor specification for the definition of Vendor Unique HBA capabilities peculiar to a particular HBA implementation.

The Asynchronous Event capabilities indicate what reasons cause the SIM to generate an asynchronous event.

This function shall return non-zero CAM Status.

- CAM Status of Request Completed Without Error indicates that the other returned fields are valid.
- CAM Status of Invalid Path ID indicates that the specified Path ID is not installed.

8.2.3 Release SIM Queue

This function is provided so that the peripheral driver can release a frozen SIM queue for the selected LUN (see 6.4.3.3).

TABLE 8-6 RELEASE SIM QUEUE

Size	Dir	Release SIM Queue
4	0	OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)

This function shall return CAM status of Request Completed Without Error.

8.2.4 Set Async Callback

This function is provided so that a peripheral driver can register a callback routine for the selected Bus/Target/LUN nexus.

TABLE 8-7 SET ASYNC CALLBACK CCB

Size	Dir	Set Async Callback
4	0	OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
4	0	Asynchronous Event Enables
		31-24 Vendor Unique
		23- 8 reserved
		7 New Devices found during rescan
		6 SIM module De-Registered
		5 SIM module Registered
		4 Sent Bus Device Reset to Target
		3 SCSI AEN
		2 reserved
		1 Unsolicited Reselection
		0 Unsolicited SCSI Bus Reset
4	0	Asynchronous Callback Pointer
4	0	Peripheral Driver Buffer Pointer
1	0	Size of Allocated Peripheral Buffer

This function shall return:

- CAM Status of Request Completed Without Error indicates that the registration of the callback routine was accepted.
- CAM Status of Request Completed with Error indicates that the registration was rejected (possibly due to invalid parameter settings).

8.2.5 Set Device Type

This function requires the XPT to add the Target ID, LUN and peripheral type to the table of attached peripherals built during CAM initialization.

78

TABLE 8-8 SET DEVICE TYPE CCB

Size	Dir	Set Device Type
4	0	- - - - - OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		- - - - - Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
1	0	Peripheral Device Type of Target/LUN

The SIM does not check the validity of the information supplied by the peripheral driver. This function shall return non-zero CAM Status.

NOTE: Blind insertion of device type information may corrupt the table, and results would be unpredictable.

- CAM Status of Request Completed Without Error indicates that the specified information was inserted into the table of SCSI devices.
- CAM Status of Request Completed with Error indicates a problem e.g. not enough room in the table to add the device information.

### 8.3 SCSI Control Functions

#### 8.3.1 Abort XPT Request

This function requests that an XPT request be aborted by identifying the CCB associated with the request. It should be issued on any I/O request that has not completed that the driver wishes to abort. Success of the Abort function is never assured. This request does not necessarily result in an Abort message being issued over SCSI.

TABLE 8-9 ABORT XPT REQUEST CCB

Size	Dir	Abort XPT Request
4	0	- - - - - OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		- - - - - Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
4	0	CCB to Be Aborted Pointer

This function shall return CAM Status of Request Completed Without Error or Unable to Abort Request.

The actual failure or success of the Abort operation is indicated by the CAM Status eventually returned in the CCB specified.

#### 8.3.2 Reset SCSI Bus

This function is used to reset the specified SCSI bus. This function should not be used in normal operation. This request shall always result in the SCSI RST signal being asserted (see 6.4.3.3 and 6.5).

TABLE 8-10 RESET SCSI BUS CCB

Size	Dir	Reset SCSI Bus
4	0	- - - - - OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		- - - - - Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)

This function shall return CAM Status of Request Completed Without Error.

The actual failure or success of the Reset SCSI Bus is indicated by the Asynchronous Callback information.

#### 8.3.3 Reset SCSI Device

This function is used to reset the specified SCSI target. This function should not be used in normal operation, but if I/O to a particular device hangs up for some reason, drivers can abort the I/O and Reset the device before trying again. This request shall always result in a Bus Device Reset message being issued over SCSI (see 6.4.3.3 and 6.5).

79

TABLE 8-11 RESET SCSI DEVICE CCB

Size	Dir	Reset SCSI Device
4	0	- - - - - OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		- - - - - Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)

This function shall return CAM Status of Request Completed Without Error.

The actual failure or success of the Reset SCSI Device is indicated by the Asynchronous Callback information.

### 8.3.4 Terminate I/O Process Request

This function requests that an XPT I/O request be terminated by identifying the CCB associated with the request. It should be issued on any I/O request that has not completed that the driver wishes to terminate. Success of the Terminate I/O Process is never assured. This request does not necessarily result in a Terminate I/O Process message being issued over SCSI.

TABLE 8-12 TERMINATE I/O PROCESS REQUEST CCB

Size	Dir	Terminate I/O Process Request
4	0	- - - - - OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		- - - - - Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
4	0	CCB to be Aborted Pointer

This function shall return CAM Status of Request Completed Without Error.

The actual failure or success of the Terminate I/O Process operation is indicated by the CAM Status eventually returned in the CCB specified.

## 9. Execute SCSI I/O

The most commonly executed request of the SIM is an I/O command, as defined in the CCB with a Function Code of Execute SCSI I/O.

### 9.1 CAM Control Block to Request I/O

Peripheral drivers should make all of their SCSI I/O requests using this function, which is designed to take advantage of all features of SCSI which can be provided by virtually any HBA/SIM combination. The CCB is as defined in Table 9-1.

This function will typically return with CAM Status of zero indicating that the request was queued successfully. Function completion can be determined by polling for non-zero status or through use of the Callback on Completion field.

TABLE 9-1 SCSI I/O REQUEST CCB

Size	Dir	SCSI I/O Request
4	0	- + - - - OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		- - - - - Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
4	0	Peripheral Driver Pointer
4	0	Next CCB Pointer
4	0	Request Mapping Information (OSD)
4	0	Callback on Completion
4	0	SG List/Data Buffer Pointer
4	0	Data Transfer Length
4	0	Sense Info Buffer Pointer
1	0	Sense Info Buffer Length
1	0	CDB Length
2	0	Number of Scatter/Gather entries
4		reserved (OSD)
1	I	SCSI Status
3		reserved (OSD)
4	I	Residual Length
12	0	CDB
		- - - - - OSD - - - - -
4	0	Timeout Value
4	0	Message Buffer Pointer (Target-only)
2	0	Message Buffer Length (Target-only)
2	0	VU Flags
1	0	Tag Queue Action
3		reserved
n	0	Private Data

9.1.1 Address of this CCB

Pointer containing the Physical address of this CCB.

9.1.2 Callback on Completion

This is an OSD field which contains the method by which the SIM is to return to the caller. In some applications it is a pointer, but in others the location of the Callback on Completion routine may be a fixed location and the CCB would contain an argument. See the OSD-specific considerations in Clause 6. The address of the Completed CCB shall be passed on the stack to inform the peripheral driver which CCB has completed.

9.1.3 CAM Control Block Length

This field contains the length in bytes of the CCB, including this field and the Address of this CCB in the total.

9.1.4 CAM Flags

This field contains bit settings as described in Table 9-2 to indicate special handling of the requested function.

TABLE 9-2 CAM FLAGS (OSD)

Size	Bits	CAM Flags (OSD)
1	7-6	Direction 00= reserved 01=In 10=Out 11=No Data Transfer
	5	1=Disable Autosense
	4	1=Scatter/Gather
	3	1=Disable Callback on Comp
	2	1=Linked CDB
	1	1=Tagged Queue Action Enable
	0	1=CDB is a Pointer
1	7	1=Disable Disconnect
	* 6	1=Initiate Synchronous Transfers
	* 5	1=Disable Synchronous Transfers
	4	SIM Queue Priority 1=Head insertion 0=Normal (tail insertion)
	3	SIM Queue Freeze
	2	Engine Synchronize
	1-0	reserved
1	7	SG List/Data 0=Host 1=Engine
	6	CDB Pointer 0=VA 1=PA
	5	SG List/Data 0=VA 1=PA
	4	Sense Buffer 0=VA 1=PA
	3	Message Buffer 0=VA 1=PA
	2	Next CCB 0=VA 1=PA
	1	Callback on Comp 0=VA 1=PA
	0	reserved
1		Target Mode-Specific CAM Flags
	7	Data Buffer Valid
	6	Status Buffer Valid
	5	Message Buffer Valid
	4	reserved
	3	1=Phase-Cognizant Mode
	2	1=Target CCB Available
	1	1=Disable AutoDisconnect
	0	1=Disable AutoSave/Restore

\* These bits are mutually exclusive

9.1.4.1 Byte 1 Bits

- 7-6 Direction - These encoded bits identify the direction of data movement during the data transfer phase, though when used in conjunction with Engine processing, they have a little different meaning (see 11).
  - a setting of 01 indicates a Read operation (data transfer from target to initiator).
  - a setting of 10 indicates a Write operation (data transfer from initiator to target).
  - a setting of 11 indicates there is to be no data transfer.
- 5 Disable Autosense - When set to 1 this bit disables autosense.
- 4 Scatter/Gather - when set to 1 this bit indicates that data is not to be transferred to/from a single location in memory but to/from several. In

18

this case the Data Buffer Pointer refers to a list of addresses and length in bytes at each address to which the data is to be transferred. The format of the SG List is defined in Table 9-3.

TABLE 9-3 SCATTER GATHER LIST

+-----+	
Size	
+-----+	
4	Data Address 1
4	Data Length 1
4	Data Address 2
4	Data Length 2
	⋮
4	Data Address n
4	Data Length n
+-----+	

- 3 Disable Callback on Completion - When set to 1 the peripheral driver does not want the SIM to callback automatically when the request is completed. This implies that the caller will be polling for a non-zero CAM Status (which indicates successful completion or termination of the request).
- 2 Linked CDB - When set to 1 this CDB is a linked command. If this bit is set, then the Control field in the CDB shall have bit 0=1. If not, the results are unpredictable. See 9.2.
- 1 Tag Queue Actions are to be enabled.
- 0 If the CDB is identified as a Pointer, the first four bytes of the CDB field contain a pointer to the location of the CDB.

9.1.4.2 Byte 2 Bits

- 7 When Disable Disconnect=1 the Disconnect capability of SCSI is disabled. The default of 0 sets bit 6=1 in the SCSI Identify MSG (which indicates that the initiator has the ability to disconnect and reconnect).
- 6 When Initiate Synchronous Transfers=1 the SIM shall negotiate Synchronous transfers, and wherever possible execute synchronous transfers.
- 5 When Disable Synchronous Transfers=1 the SIM shall negotiate Asynchronous transfers (if previously negotiated Synchronous). If unable to negotiate Synchronous or negotiation has not yet been attempted, the SIM shall not initiate negotiation.
- 4 When SIM Queue Priority=1 the SIM shall place this CCB at the head of the Target/LUN internal queue to be the next operation sent to the Target/LUN by the SIM.
- 3 When SIM Queue Freeze=1 the SIM shall place its internal Target/LUN queue into the frozen state. Upon callback, the CAM Status for this CCB shall have the SIM Queue Freeze flag set. This bit should only be set for SIM error recovery and should be used in conjunction with the SIM Queue Priority bit and the Release SIM Queue command.
- 2 The Engine Synchronize=1 is used in conjunction with the In or Out setting to flush any residual bits before terminating engine processing (see 11).

9.1.4.3 Byte 3 Bits

The Pointer fields are set up to have one characteristic. If a bit is set to 1 it means the pointer contains a Physical Address. If set to 0 it means the pointer contains a Virtual Address. If the SIM needs an address in a different form to that provided, it should be converted by the SIM (using OSD facilities) and stored in Private Data.

9.1.4.4 Byte 4 Bits

The Target Mode Only Flags are only active on Enable LUN or Execute Target I/O commands.

- 7-5 The Buffer Valid bits identify which buffers have contents. In the event that more than one bit is set, they shall be transferred in the sequence of Data Buffer, Status, Message Buffer.
- 3 Phase-Cognizant Mode - if target operations are supported, when set to 1, the SIM shall operate in Phase-Cognizant Mode, otherwise it shall operate in Processor Mode.
- 2 Target CCB Available - when set to 1 this bit indicates that the XPT/SIM can use this CCB to process this request. A value of 0 indicates that this CCB is not available to the XPT/SIM.
- 1 AutoDisconnect - when set to 1 this bit disables AutoDisconnect. The default of 0 causes the XPT/SIM to automatically disconnect, if the Identify message indicates DiscPriv is set.
- 0 AutoSave - when set to 1 this bit disables AutoSave. The default of 0 causes the XPT/SIM to automatically to send a Save Data Pointer message on an AutoDisconnect.

9.1.5 CAM Status

This field is returned by the SIM after the function is completed. A zero status indicates that the request is still in progress or queued. CAM Status is defined in Table 9-4.

If Autosense information is available, the code returned shall be incremented by 80h e.g. 04h indicates an error occurred, and 84h indicates that an error occurred and Autosense information is available for analysis.



TABLE 9-4 CAM STATUS

00h	Request in progress
01h	Request completed without error
02h	Request aborted by host
03h	Unable to Abort Request
04h	Request completed with error
05h	CAM Busy
06h	Invalid Request
07h	Invalid Path ID
08h	SCSI device not installed
09h	Unable to Terminate I/O Process
0Ah	Target Selection Timeout
0Bh	Command Timeout
0Ch	reserved
0Dh	Message Reject received
0Eh	SCSI Bus Reset Sent/Received
0Fh	Uncorrectable Parity Error Detected
10h	Autosense Request Sense Cmd Failed
11h	No HBA detected
12h	Data OverRun/UnderRun
13h	Unexpected Bus Free
14h	Target bus phase sequence failure
15h	CCB Length Inadequate
16h	Cannot Provide Requested Capability
17h	Bus Device Reset Sent
18h	Terminate I/O Process
19-37h	reserved
Target Mode Only Status	
38h	Invalid LUN
39h	Invalid Target ID
3Ah	Function not Implemented
3Bh	Nexus not Established
3Ch	Invalid Initiator ID
3Dh	SCSI CDB Received
3Eh	LUN Already Enabled
3Fh	SCSI bus Busy
+40h	to indicate that SIM Queue is frozen
+80h	to indicate that Autosense is valid

- 00h Request in progress: the request is still in process.
- 01h Request completed without error: the request has completed and no error condition was encountered.
- 02h Request aborted by host: the request was aborted by the peripheral driver.
- 03h Unable to Abort Request: the SIM was unable to abort the request as instructed by the peripheral driver.
- 04h Request completed with error: the request has completed and an error condition was encountered.
- 05h CAM Busy: CAM unable to accept request at this time.
- 06h Invalid Request: the request has been rejected because it is invalid.
- 07h Invalid Path ID indicates that the Path ID is invalid.
- 08h SCSI device not installed: Peripheral Device Type field is not valid.
- 09h Unable to Terminate I/O Process: the SIM was unable to terminate the request as instructed by the peripheral driver.

- 0Ah Target Selection Timeout: The target failed to respond to selection.
- 0Bh Command Timeout: the specified command did not complete within the timer value specified in the CCB.
- 0Dh Message Reject received: The SIM received a Message Reject MSG.
- 0Eh SCSI Bus Reset Sent/Received: The SCSI operation was terminated at some point because the SCSI bus was reset.
- 0Fh Uncorrectable Parity Error Detected: An uncorrectable SCSI bus parity error was detected. When this occurs, the SIM sends the ABORT message to the target.
- 10h Autosense Request Sense Command Failed: The SIM attempted to obtain sense data and failed.
- 11h No HBA detected: HBA no longer responding to SIM (assumed to be a hardware problem).
- 12h Data OverRun: target transferred more data bytes than peripheral driver indicated in the CCB.
- 13h Unexpected Bus Free: an unexpected Bus Free condition occurred.
- 14h Target Bus Phase Sequence Failure: the target failed to operate in a proper manner according to X3.131-1990 e.g. it went to the Message Out phase after the initiator asserted ATN.
- 15h CCB Length Inadequate: More private data area is required in the CCB.
- 16h Cannot Provide Requested Capability: Resources are not available to provide the capability requested (in the CAM Flags).
- 17h Bus Device Reset Sent: this CCB was terminated because a Bus Device Reset was sent to the target.
- 18h Terminate I/O Process: this CCB was terminated because a Terminate I/O Process was sent to the target.
- 38h Invalid LUN indicates that the LUN specified is outside the supported range of the SCSI bus.
- 39h Invalid Target ID indicates that the Target ID does not match that used by the HBA specified by the Path ID field.
- 3Ah Function Not Implemented indicates that Target Mode is not supported.
- 3Bh Nexus not Established: There is currently no connection established between the specified Target ID and Target LUN and any initiator.
- 3Ch Invalid Initiator ID: The initiator ID specified is outside the valid range that is supported.

NOTE: This status can also be returned if the target tries to reselect an initiator other than the one to which it was previously connected.

- 3Dh SCSI CDB Received: Indicates that the target has been selected and that the SCSI CDB is present in the CCB.
- 3Eh LUN Already Enabled: The LUN identified in Enable LUN was previously enabled.
- 3Fh SCSI bus Busy: The SIM failed to win arbitration for the SCSI Bus during several different bus free phases.

9.1.6 CDB

This field either contains the SCSI CDB (Command Descriptor Block), or a pointer to the CDB, to be dispatched.

9.1.7 CDB Length

This field contains the length in bytes of the CDB.

9.1.8 Data Transfer Length

This field contains the length in bytes of the data to be transferred.

83

9.1.9 Function Code

See 8.1.2.

9.1.10 LUN

This field identifies the SCSI LUN to which this CCB is being directed.

9.1.11 Message Buffer Length (Target-only)

This field contains the length in bytes of the field which is to be used to hold Message information in the event that the Peripheral Drivers needs to issue any MSGs. This field is exclusive to Target Mode operation.

9.1.12 Message Buffer Pointer (Target-only)

This field contains a pointer to buffer containing Messages. This pointer is only valid for use in Target Mode.

9.1.13 Next CCB Pointer

This field contains a pointer to the next command block in a chain of command blocks. A value of 0 indicates the last command block on the chain. This field is used for the linking of commands.

9.1.14 Number of Scatter/Gather entries

This field contains the number of entries in the SG List.

9.1.15 Path ID

The Path ID specifies the SCSI port on the installed HBA to which the request is addressed. Path IDs are assigned by the XPT, begin with zero, and need not be consecutive. The Path ID of FFh is assigned for the XPT. An HBA may have more than one SCSI port. A SIM may support more than one HBA.

9.1.16 Peripheral Driver Pointer

This field contains a pointer which is for the exclusive use of the Peripheral Driver, which use is not defined by this standard.

9.1.17 Private Data

This field is used to contain whatever fields the CAM Module needs to execute the request. As such it constitutes a scratchpad of working space needed by the SIM and/or the XPT. The size of this area is an OSD as it may differ between SIMs and XPTs by environment or by vendor implementation. The device driver is responsible to query the XPT and ensure that enough Private Data area is available to the SIM and/or XPT.

9.1.18 Request Mapping Information (OSD)

This field is a pointer to an OSD dependent data structure which is associated with the original I/O request.

9.1.19 Residual Length

This field contains the difference in twos complement form of the number of data bytes transferred by the HBA compared with the number of bytes requested by the CCB.

9.1.20 SCSI Status

This field contains the status byte returned by the SCSI target after the command is completed.

9.1.21 Sense Info Buffer Length

This field contains the length in bytes of the field which is to be used to hold Sense data in the event that a Request Sense is issued.

9.1.22 Sense Info Buffer Pointer

This field contains a pointer to the data buffer for Request Sense data. This pointer will only be used if a Check Condition occurs while performing the specified command.

9.1.23 SG List/Data Buffer Pointer

This field contains a pointer to either the data buffer to which data is to be transferred, or to the SG List which contains the list of scatter/gather addresses to be used for the transfer.

9.1.24 Tagged Queue Action

SCSI provides the capability of tagging commands to force execution in a specific sequence, or of letting the target optimize the sequence of execution to improve performance. This function provides a similar capability. For a description of the tagged command queueing philosophy see SCSI-2 X3.131-1990.

When the Queue Action Enable bit in the CAM Flags is set, the CDB issued by the SIM shall be associated with the Queue Action specified as:

- 20h = Simple Tag Request
- 21h = Head of Queue Tag Request
- 22h = Ordered Queue Tag Request

9.1.25 Target ID

This field identifies the SCSI target which is to be selected for execution of the CCB request.

9.1.26 Timeout Value

This field contains the maximum period in seconds that a request can remain outstanding. If this value is exceeded then the CAM Status shall report the timeout condition. A value of 00h in the CCB means the peripheral driver accepts the SIM default timeout. A value of F...Fh in the CCB specifies an infinite period.

78

9.1.27 VU Flags

The uses for this field are defined in the vendor specification.

9.2 Command Linking

The SIM supports SCSI's ability to link commands in order to guarantee the sequential execution of several requests. This function requires that both the HBA and the involved target(s) support the SCSI Link capability.

To utilize linking, a chain of CCBs is built with the Next CCB Pointer being used to link the CCBs together. The CAM Flag Link bit shall be set in all CCBs but the last in the chain. When a SCSI target returns the Linked Command Complete message, the next CCB is processed, and its associated CDB is dispatched.

Any Check Condition returned by the target on a linked command shall break the chain.

10. Target Mode (Optional)

If a Target Mode function is specified by a CCB and this functionality is not provided by a particular SIM implementation, then a CAM Status of Function Not Implemented shall be returned in the CCB.

The Target Mode functionality causes the HBA associated with the specified SCSI link to be set up so that it may be selected as a target i.e. when an HBA is operating in Target mode, it is responding to other HBAs on the same SCSI cable.

There are two different modes of target operation, either or both of which may be supported by the XPT/SIM as defined by the Target Mode Support flags in the Path Inquiry CCB.

- Processor mode
- Phase-Cognizant mode

Processor mode permits an application to register itself as a LUN and provide a set of one or more CCBs that the XPT/SIM can use for receiving and sending data. In this mode, when the adapter is selected and the XPT/SIM receives an Identify message for a LUN that has registered as a Processor LUN, the XPT/SIM will accept any processor device commands (Inquiry, Request Sense, Send, Receive) and, using one of the available CCB's, process the SCSI command through completion.

Upon disconnection, the SIM calls back on completion to let the application know that the CCB has been processed. From the time that the application registers itself until the time a command has completed, there is no callback to the application.

In summary, Processor applications get called back only after the SCSI command has been completely processed, and leaves all phase handling and SCSI command processing nuances to the XPT/SIM and the previously registered CCB's.

Phase-Cognizant mode permits an application tighter control over what takes place when a SCSI command is received by the SIM. When a Phase-Cognizant application registers itself and a command is received, the XPT/SIM does an

immediate Callback on Completion after placing the SCSI command in an available CCB. The Phase-Cognizant application is responsible to set up data, message, status fields and CAM-Flags in the CCB and re-issue the CCB with an Execute Target I/O function code so that the XPT/SIM knows which phases it should execute. The "callback-reissue CCB" cycle may happen multiple times before a command completes execution.

In summary, Phase Cognizant applications get a callback immediately after the SCSI command block is received and is expected to instruct the XPT/SIM as to which phases to go through to perform the command.

10.1 Enable LUN

The specified Target ID shall match that returned by the HBA Inquiry Function for the HBA. The specified LUN is the one enabled for selection, and if the HBA is to respond as an additional LUN, another Enable LUN is required.

In addition to providing a hook into the application, this function is intended to provide an area that the XPT/SIM can use as working space when the HBA is selected.

TABLE 10-1 ENABLE LUN CCB

Size	Dir	Enable LUN CCB
4	0	OSD Address of this CCB
2	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
1	0	Common
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
2	0	Group 6 Vendor Unique CDB Length
2	0	Group 7 Vendor Unique CDB Length
4	0	Pointer to Target CCB List
2	0	Number of Target CCBs

If the Number of Target CCBs is zero, then Target Mode is disabled, otherwise the Pointer to Target CCB List refers to a list of addresses of CCBs to which the data is to be transferred (see Table 10-2).

TABLE 10-2 TARGET CCB LIST

Size	Target CCB List
4	CCB Address 1
4	CCB Address 2
	:
4	CCB Address n

The XPT/SIM shall place the pointer to the CCB, or the pointer to the list of

85

CCBs, in a list until the specified Target ID and LUN is selected on the SCSI link specified by the Path ID field. While the request is being held, the CAM Status field of the Target CCB, shall be set to Request in Progress. The application is required to poll on the CAM status field of the Target CCB or provide a Completion Callback routine through the Target CCB.

The XPT/SIM shall keep an indication of whether a single CCB or list of CCBs was provided on the Enable LUN service.

The XPT/SIM shall set the following in each Target CCB when they are first provided:

- CAM Status to Request In Progress
- CAM Flags shall be the same as those in the Enable LUN CCB
- CAM Flags shall set the Target CCB Available as needed

Within the Target CCB provided, the following information shall be present and valid,

- CAM Flag information including AutoDisconnect and AutoSave.
- CDB field is valid for the Command Blocks that may be received. That is either CDBs are embedded in the CCB, or a pointer to a CDB area is provided in the CDB field.
- The Group 6 and 7 Vendor Unique CDB Length fields contain the number of bytes a target application expects to receive for it's vendor unique command set. The previous item shall go hand-in-hand with this requirement. The Group 6 and 7 Vendor Unique CDB Length fields shall be retained for each LUN enabled.

If the target application supports Vendor Unique Command Blocks, then the CDB field of the CCB shall reflect the nature and size of those Vendor Unique Command Blocks. Ample space shall be provided to contain the CDBs that may be received. If a CDB greater than the size of the CDB field is desired, then the CDB field shall contain a pointer to a CDB.

To disable the selection of a specific LUN, the application performs an Enable LUN with a zero value for the Number of Target CCBs.

If a LUN is disabled, after having been enabled, then the Inquiry data and the Vendor Unique CDB Length data shall be cleared.

The XPT/SIM shall prevent a nexus being established between an initiator and a specified LUN that has been disabled. If there is a pre-existing nexus, then Invalid Request shall be returned.

This function shall return non-zero CAM Status.

- CAM Status of Request Completed Without Error indicates that the Enable LUN was completed successfully.
- CAM Status of Invalid Request indicates that there is currently a nexus established with an initiator that shall be terminated, first.
- CAM Status of Invalid Path ID indicates that the Path ID is invalid.
- CAM Status of Invalid Target ID indicates that the Target ID does not match that used by the HBA specified by the Path ID field.
- CAM Status of Invalid LUN indicates that the LUN specified is outside the supported range of the SCSI bus.
- CAM Status of Function Not Implemented indicates that Target Mode is not

supported by this implementation of CAM.

## 10.2 Phase Cognizant Mode

### 10.2.1 Target Operation of the HBA

When the HBA is selected, the XPT/SIM automatically sets the HBA to the Message Out phase to receive the Identify, Synchronous Data, and other messages that may be sent by the Initiator. The XPT/SIM response to these messages shall be as defined in X3.131-1990.

The LUN shall be extracted from the Identify Message and the appropriate CCB shall be extracted from the list of CCBs being held by the XPT/SIM.

If the LUNTAR bit (or any of the reserved bits) of the Identify Message is set to 1, then the XPT/SIM shall send a MESSAGE REJECT message back to the initiator.

If no CCBs are being held by the XPT/SIM for a Target ID, then the XPT/SIM shall not respond to the selection of that Target ID.

If CCBs are being held by the XPT/SIM, and the LUN indicated by the Identify Message does not have a CCB provided by an application, then the XPT/SIM shall provide the following support:

- a) If an Inquiry command is sent to this LUN, then the XPT/SIM shall respond with Inquiry Data that indicates "Logical Unit Not Supported."
- b) If any other command (except Request Sense) is sent to this LUN, then the XPT/SIM shall respond with a Check Condition.
- c) If a Request Sense command is sent to this LUN after a Check Condition status is sent, then the XPT/SIM shall respond with sense data that indicates "Logical Unit Not Supported".

The XPT/SIM shall scan the CAM Flags in the CCB(s) provided with Enable LUN. If none of them have the Target CCB Available bit set, the XPT/SIM shall request the SCSI CDB and post BSY status. The XPT/SIM shall not modify the SCSI CDB(s) in the CCB(s).

After processing the CDB from a Target CCB, the target application shall set CCB Available in the CAM Flags, which allows the application to pass the same CCB back to the XPT/SIM upon Callback on Completion (this prevents the possibility that the XPT/SIM could use the CCB on selection). The setting of the Target Available bit could be done at the Callback on Completion after the Execute Target I/O which transmits SCSI Status.

If a target application sets Target Available upon recognizing that a CDB has been received and uses a different CCB to perform the data transfer, there is a lower likelihood of a BSY response to the initiator when a CCB is not available.

The Disable Disconnect bit in the CAM Flags field shall be updated to indicate the state of the DiscPriv bit in the Identify message that was received from the initiator. If the DiscPriv bit was set in the Identify Message, then the Disable Disconnect bit shall be cleared, and vice-versa. NOTE: The default state of the Disable Disconnect bit in the CAM Flags is cleared, implying that disconnect is enabled.

The Target ID field shall be set to the ID of the initiator that performed the selection. This field can then be used by subsequent functions, such as reselect, to determine the Initiator's ID.

Once the initial Message Out Phase is complete, the XPT/SIM automatically sets the HBA to the Command Out Phase to request the SCSI CDB. After receiving the SCSI CDB bytes, the XPT/SIM shall set the CAM status field to CAM Status of SCSI CDB received, and clear the CCB Available bit in the CAM Flags.

Upon completion of the data phase, the XPT/SIM shall send the appropriate SCSI status and Command Complete and then disconnect from the bus. The XPT/SIM shall then post the required CAM Status in the CCB, or Callback on Completion.

If the Group Code of the Operation Code of the Command Block is Vendor Unique the XPT/SIM shall ensure that only the indicated number of command bytes are received. If the required number of bytes are exceeded or not transferred, then the XPT/SIM shall return a status of Check Condition, the Sense Key in the Sense Buffer shall be set to Illegal Request, and the Additional Sense Key and Qualifier shall be set to Command Phase Error.

If the DiscPriv bit in the Identify message was set, which results in the Disable Disconnect bit of the CAM Flags being cleared, and the Disable AutoDisconnect bit of the CAM Flags field is cleared, the XPT/SIM shall automatically disconnect upon receipt of the command block. The subsequent invocation of the Execute Target I/O function shall perform an automatic reselect when it is invoked.

If a BUS DEVICE RESET message is received at any time, the XPT/SIM shall set the CAM Status field to SCSI Bus Reset Sent/Received for any CCB being held (through Enable LUN), or that is active in the XPT/SIM.

If a SCSI Bus Reset occurs the asynchronous callback and bus reset mechanism defined for initiator mode shall be followed.

The SIM shall reject any CCB which has a Timeout Value of other than infinity.

### 10.2.2 Execute Target I/O

If the Data Valid bit is set, the XPT/SIM shall enter the data phase indicated by the direction bit in the CAM Flags field (ie. DATA IN or DATA OUT). It shall send/receive data to/from the buffer(s) indicated in the CCBs Scatter Gather List or Data Pointer.

If the Status Valid bit is set, the XPT shall send the status byte specified in the SCSI Status field to the current initiator and then send the Command Complete Message.

If the Message Valid bit is set, the XPT shall enter the Message phase and transfer the contents of the Message buffer.

The XPT/SIM shall receive and respond to any messages resulting from ATN being asserted by the initiator, in addition to any messages it sends to the initiator.

The XPT/SIM shall be able to execute all the phases indicated by the Buffer

Valid bits of the CAM Flags, within a single invocation of the Execute Target I/O i.e. if more than one bit is set, the order of execution of the phases shall be data, status, and message.

If the Data Buffer Valid and Status Buffer Valid bits of the CAM Flags are both set for an invocation of Execute Target I/O, the AutoDisconnect and AutoSave features shall be disabled.

If the Disable AutoDisconnect bit of the CAM Flags is cleared, and the Disable Disconnect of the CAM Flags bit is cleared, then the XPT/SIM shall disconnect on the completion of the data transfer.

If the Disable AutoSave bit of the CAM Flags is cleared, then the XPT/SIM shall send a Save Data Pointers message to the initiator prior to disconnect.

The XPT/SIM shall perform an automatic reselect if the XPT/SIM had disconnected after the receipt of the CDB, or had disconnected upon completion of a previous Execute Target I/O (within the same I/O process).

Upon the last Execute Target I/O, the target application should consider setting the Disable AutoSave bit, which shall disable the sending of the Save Data Pointers.

This function typically returns with CAM Status of zero indicating that the request was executed successfully. Function completion can be determined by polling for non-zero status or through use of the Callback on Completion field.

## 10.3 Processor Mode

### 10.3.1 CCB Acceptance

In Processor mode, the Target CCB List shall contain at least one pre-built CCB that the SIM can use when it responds to selection. The Target CCBs that are supported by the SIM include CDBs for the following commands:

- Inquiry
- Receive
- Request Sense
- Send

The SIM shall verify that the CCBs in the Target CCB List contain supported commands, valid data buffers etc.

Any invalid CCB in the list shall be rejected and the LUN shall not be enabled.

### 10.3.2 Target Operation of the HBA

When the target HBA is selected, it shall automatically request the CDB.

The SIM shall search the Target CCB List to find a matching CDB. If a matching CDB is found, it shall verify that Target CCB Available=1, and use the contents of the data buffers to process the command received. The SIM shall clear Target CCB Available, and if the peripheral driver wants the CCB to be re-used it is responsible to set Target CCB Available=1.

87

Upon completion of the CDB received, the SIM shall report CAM Status in the CCB and call back the peripheral driver.

If the Target CCB List has no CCBs with Target CCB Available=1, but matches were found, the SIM shall send Busy Status to the Initiator.

If the Target CCB List contained no matching CCBs, then the SIM shall return Check Condition to the Initiator. Upon receipt of a Request Sense command, the SIM shall return a Sense code of "Invalid CDB" to the Initiator.

If an Inquiry CDB is received but there is no Inquiry CDB in one of the CCBs in the Target CCB List then the SIM shall return Inquiry Data of "LUN Not Supported" to the Initiator. NOTE: A CCB to respond to an Inquiry CDB should be provided in every Target CCB List.

If an Inquiry CDB is and there is an Inquiry CDB in one of the CCBs in the Target CCB List then the SIM shall return the Inquiry information provided by the data buffer pointer. The SIM does not clear Target CCB Available or call back as it is a placeholder of consistent information.

### 11. HBA Engines

An engine is a hardware device implemented in an HBA to perform time-intensive functions not available on target devices. Generally, these engines are required to process data prior to building a CDB and submitting to the device. There may be more than one engine in a HBA.

One use of engines is to compress data. In this mode, a device driver first submits data to the engine. Once the engine has completed processing the data, an Execute SCSI CCB can be built for the SCSI transfer.

The engine model allows for the addressing of buffer memory located on the HBA. The buffer addressing appears to the host as contiguous space. Using this model, it is possible to submit multiple requests until the engine buffer is full. Once the full condition is met, an Execute SCSI CCB can be built.

When the full condition occurs (as defined by the Destination Data Length equalling the Destination Data Maximum Length), the amount of unprocessed source data is reported in the Source Residual Length. The residual data may then be re-submitted at a later time.

#### 11.1 Engine Inquiry

This function is used to learn information on the engines installed in the HBA hardware. This function is issued for each engine.

TABLE 11-1 ENGINE INQUIRY CCB

Size	Dir	Engine Inquiry
4	0	OSD - - - - -
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		- - - - - Common - - - - -
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
2	0	Engine Number
1	I	Engine Type
		0=Buffer Memory
		1=Lossless Compression
		2=Lossy Compression
		3=Encryption
		4=FF reserved
1	I	Engine Algorithm ID
		0=Vendor Unique
		1=LZ1 Variation 1 (STAC)
		2=LZ2 Variation 1 (HP DCZL)
		3=LZ2 Variation 2 (Infochip)
		4=FF reserved
4	I	Engine Memory Size

The Engine Type reports the generic function the addressed engine number is capable of supporting.

The Engine Algorithm ID reports the specific capability the addressed engine supports.

The amount of buffer memory provided for an engine is reported in the Engine Memory Size.

This function shall return non-zero CAM Status.

- CAM Status of Request Completed Without Error indicates that the other returned fields are valid.
- CAM Status of Invalid Request indicates that the specified Engine Number is not installed.

#### 11.2 Execute Engine Request (Optional)

To accommodate buffering associated with the engine, the CAM Flag SG List/Data set to 1=Engine is used to specify that the normal Data Buffer Pointer is actually a physical address in the buffer space of the engine.

There are four modes associated with engine processing established by CAM Flags:

- A Direction setting of Out is used to Encrypt or Compress the data
- A Direction setting of Out is used to Decrypt or Decompress the data

- Synchronize is used in conjunction with In or Out to flush any residual bits prior to terminating engine processing.

Certain types of engines change the size of data as part of their operation e.g. the purpose of compression engines is to reduce the size of data prior to transmission over SCSI. As such, the Execute Engine Request CCB allows the engine to report the size of the resultant data.

TABLE 11-1 EXECUTE ENGINE REQUEST CCB

Size	Dir	Execute Engine Request
4	0	OSD
2	0	Address of this CCB
1	0	CAM Control Block Length
1	0	Function Code
1	I	CAM Status
1		reserved
		Common
1	0	Path ID
1	0	Target ID
1	0	LUN
4	0	CAM Flags (OSD)
4	0	Peripheral Driver Pointer
4	0	reserved (OSD)
4	0	Request Mapping Information (OSD)
4	0	Callback on Completion
4	0	SG List/Data Buffer Pointer
4	0	Data Transfer Length
4	0	Engine Buffer Data Pointer
1		reserved (OSD)
1		reserved (OSD)
2	0	Number of Scatter/Gather entries
4	0	Destination Data Maximum Length
4	I	Destination Data Length
4	I	Source Residual Length
12		reserved (OSD)
		OSD
4	0	Timeout Value
4		reserved
2	0	Engine Number
2	0	VU Flags
1		reserved
3		reserved
n	0	Private Data

This function will typically return with CAM status of zero indicating that the request was queued successfully. Function completion can be determined by polling for non-zero status or through use of the Callback on Completion field.

ANNEX

Annex A. Physical/Logical Translation in 80x86 Environment

A.1 OSD Formatting of Disk Drives

The DOS physical address to/from logical block address conversion algorithms to map SCSI disks into int 13h Head-Cylinder-Sector format vary widely between suppliers of software to support third party disks.

The following "C" routines have been adopted by CAM as representing the most efficient utilization of capacity. The following code is ANSI "C" that can be compiled using the Microsoft C compiler, version 5.1.

- a) SETSIZE converts a Read Capacity value to int 13h Head-Cylinder-Sector requirements. It minimizes the value for number of heads and maximizes the number of cylinders. This will support rather large disks before the number of heads will not fit in 4 bits (or 6 bits). This algorithm also minimizes the number of sectors that will be unused at the end of the disk while allowing for very large disks to be accommodated. This algorithm does not use physical geometry.
- b) LTOP does logical to physical conversion
- c) PTOL does physical to logical conversion
- d) MAIN is a test routine for a, b and c.

A.1.1 SETSIZE

```

/*
typedef unsigned int UINT;
typedef unsigned long ULNG;
/*
* Convert from logical block count to Cylinder, Sector and Head (int 13)
*/

int setsize(ULNG capacity, UINT *cyls, UINT *hds, UINT *secs)
{
    UINT rv = 0;
    ULNG heads, sectors, cylinders, temp;

    cylinders = 1024L; /* Set number of cylinders to max value */
    sectors = 62L; /* Max out number of sectors per track */

    temp = cylinders * sectors; /* Compute divisor for heads */
    heads = capacity / temp; /* Compute value for number of heads */
    if (capacity % temp) { /* If no remainder, done! */
        heads++; /* Else, increment number of heads */
        temp = cylinders * heads; /* Compute divisor for sectors */
        sectors = capacity / temp; /* Compute value for sectors per track */
        if (capacity % temp) { /* If no remainder, done! */
            sectors++; /* Else, increment number of sectors */
            temp = heads * sectors; /* Compute divisor for cylinders */
            cylinders = capacity / temp; /* Compute number of cylinders */
        }
    }
}
    
```

69

```

    }
  }
  if (cylinders == 0) rv=1;      /* Give error if 0 cylinders */
  *cyls = (UINT) cylinders;     /* Stuff return values */
  *secs = (UINT) sectors;
  *hds = (UINT) heads;
  return(rv);
}

```

A.1.2 LTOP

```

/*
 * logical to physical conversion
 */
void ltop(ULNG block,UINT hd_count,UINT sec_count,UINT *cyl,UINT *hd,UINT
*sec)
{
  UINT spc;
  spc = hd_count * sec_count;
  *cyl = block / spc;
  *hd = (block % spc) / sec_count;
  *sec = (block % spc) % sec_count;
}

```

A.1.3 PTOL

```

/*
 * Physical to logical conversion
 */
ULNG ptol(UINT cyl,UINT hd,UINT sec,UINT cyl_count,UINT hd_count,UINT
sec_count)
{
  ULNG cylsize;
  cylsize = sec_count * hd_count;
  return((cyl * cylsize) + (hd * sec_count) + sec);
}

```

A.2 Backwards Compatibility

The selection of a new algorithm for CAM solves the problem of future compatibility, but it does not solve the problem of the installed base. The following technique will permit a supplier to update the installed base to CAM-compliant operation but not require users to reformat their drives.

A.2.1 ROM-based

The one sector that is independent of the algorithm is sector 00. Under DOS and many other Operating Systems this sector is used for the boot sector and contains the Partition Table for a fixed disk.

If the Partition Table is structured according to MS DOS and IBM DOS standards, partitions end on cylinder boundaries e.g.

Offset from start of Partition	Table entry
00h	Boot Indicator
01h	Beginning or start head
02h	beginning or start sector
03h	Beginning or start cylinder
04h	System indicator
05h	Ending head
06h	Ending sector
07h	Ending cylinder
08h	Starting sector (relative to beginning of disk)
0Ch	Number of sectors in partition

The ending head 07h indicates a device with 8 heads (0 to 7). The ending sector 91h contains 2 bits of high cylinder so it has to be masked to obtain ending sector = 11h (17 decimal).

To verify these values calculate:

Logical Ending sector (from Beginning Head, Cylinder, and Sector)

and compare it to:

(Starting Sector + Number of Sectors in Partition)

This leaves Number of Cylinders as the one unresolved parameter. This is obtained by:

Read Capacity divided by (Heads \* Sectors).

All of this can be done by the BIOS in ROM or RAM. To be capable of booting from any drive or cartridge regardless of the algorithm used to partition and format the media, the BIOS would need to respond to int 13 function 8 with the head, sector, and cylinder values obtained from this information. In addition, the BIOS would need to use those values in its calculation from physical to logical sectors.

Example of Pseudocode:

```

For each Drive
  Read Boot Sector (LBA 0)
  Validate The Signature at end of Sector (55AA)
  Find Partition with largest Logical Start Cyl

  If No Partitions found
    Use Defaults
    Exit

  SECS = Ending Sector (from partition table)
  Heads = Ending Head+1 (from partition table)

  Logical_End = End_cyl * (End_head+1 * End_sector) +
               (End_head * End_sector) + End_sector

  Compare Logical_End to Starting_sec + Number_sec
  If not equal

```



Use Defaults  
Exit

Cyls = Capacity / (End\_head+1 \* End\_sector)

A.2.2 RAM-based

Under DOS it is possible to modify the code of the boot sector to accomplish bootability. Access to other partitions is dependent on the device driver to do a translation.

This method is a patch just prior to jumping to code loaded in memory at segment 00 offset 7C00h.

```

PUSH AX      ; Save registers used in patch
PUSH DX
MOV AH,08   ; set function code = 8 get drive parameters
INT 13      ; do INT 13 call
INC DH      ; inc head number to convert from zero based
MOV [7C1A],DH ; fix value of heads in BPB table
AND CL,3F   ; Mask off non-sector information
MOV [7C18],CL ; fix value of sectors in BPB table
POP DX
POP AX      ; Restore registers used in patch
JMP 7C00    ; jump to partition boot loader
    
```

```

01B0 00 00 00 00 00 00 00 00-00 00 00 00 00 80 01
01C0 01 00 06 07 91 7C 11 00-00 00 57 52 01 00 00 00
01D0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
01E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00
01F0 00 00 00 00 00 00 00 00-00 00 00 00 00 55 AA
    
```

16

7C00	7C03	7C0B	7C0D	7C0E
jump nop	Name I B M 4 . 0	Bytes/ Sector	Sectors/ Cluster	Reserved Sectors
EB 3C 90	49 42 4D 20 20-34 2E 30	00 02	04	01 00

7C10	7C11	7C13	7C15	7C16	7C18	7C1A
# FATs 02	# DIR entries 00 02	# Log'l Sectors 00 00	Media Descrip F8	# FAT Sectors 55 00	# Sectors 11 00	# Heads 08 00

Annex B: Target Application Examples

[ Mike Roche to prepare an update which is consistent with Section 10.]

The following are examples of how a Target Application can operate the Target Mode capabilities defined in Section 9.

B.1 Initialization Sequence with Single Target CCB provided

- fill Target CCB with required info  
targetCCB.callbackPointer = callback routine address
- fill Enable CCB with the required information  
enableCCB.functionCode = function code for enable lun  
enableCCB.targetid = the id of the target  
enableCCB.targetLun = the lun to enable  
enableCCB.group6VULength = vendor unique length for Group 6 (IF required)  
enableCCB.group7VULength = vendor unique length for Group 7 (IF required)  
enableCCB.targetCCBListLength = 0  
enableCb.targetCCBPointer = &targetCCB
- Enable LUN (&enableCCB)
- EXIT

B.2 Initialization Sequence with Multiple Target CCBs provided

- fill Target CCB #1 with required info  
target1CCB.callbackPointer = callback routine address #1
- fill Target CCB #2 with required info  
target2CCB.callbackPointer = callback routine address #2  
target2CCB.camStatus = request completed by target application
- fill Target CCB #n with required info  
targetnCCB.callbackPointer = callback routine address #n  
targetnCCB.camStatus = request completed by target application
- targetCCBList [0] = pointer to target1CCB
- targetCCBList [1] = pointer to target2CCB
- targetCCBList [n] = pointer to targetnCCB

NOTE: where targetCCBList is an array of pointers

- fill enable CCB with the required information  
enableCCB.functionCode = function code for enable lun  
enableCCB.targetid = the id of the target  
enableCCB.targetLun = the lun to enable  
enableCCB.group6VULength = vendor unique length for Group 6 (IF required)  
enableCCB.group7VULength = vendor unique length for Group 7 (IF required)  
enableCCB.targetCCBListLength = 4 \* number of target CCBs  
enableCb.targetCCBPointer = &targetCCBList
- Enable LUN (&enableCCB)
- EXIT

B.3 Application Sequence with single Execute Target I/O

- IF targetCCB.camStatus != SCSI CDB Received THEN EXIT, OR  
callback from XPT/SIM
- process SCSI CDB field in targetCCB
- fill targetCCB with required information  
targetCCB.functionCode = function code for execute target io  
targetCCB.camFlags = data phase and status phase

```

targetCCB.dataBufferPointerLength = length of data
targetCCB.dataBufferPointer = pointer to data buffer
targetCCB.SCSIStatus = whatever status is appropriate
- Execute Target IO (&targetCCB)
/* return target CCB to pool */
- targetCCB.camStatus = request in progress
- EXIT

```

## B.4 Application Sequence with multiple Execute Target I/O

```

- IF targetCCB.camStatus != SCSI CDB Received THEN EXIT, OR
  callback from XPT/SIM
- process SCSI CDB field in targetCCB
- loop until all data transferred
  fill targetCCB with required information
  targetCCB.functionCode = function code for execute target io
  targetCCB.camFlags = data phase
  targetCCB.dataBufferPointerLength = length of data
  targetCCB.dataBufferPointer = pointer to data buffer

  IF (last data block)
    targetCCB.camFlags = data phase AND status phase
    targetCCB.SCSIStatus = whatever status is appropriate

    Execute Target IO (&targetCCB)
  - end loop
/* return target CCB to pool */
- targetCCB.camStatus = request in progress
- EXIT

```

## Annex C: Unix OSD Data Structures

```

/* ----- */
/* cam.h Version 1.05 Nov. 07, 1990 */
/* This file contains the definitions and data structures for the CAM
  Subsystem interface. The contents of this file should match the
  data structures and constants that were specified in the CAM document,
  CAM/89-003 Rev 2.2.
*/
/* ----- */
/* Defines for the XPT function codes, Table 8-2 in the CAM spec. */
/* Common function commands, 0x00 - 0x0F */
#define XPT_NOOP 0x00 /* Execute Nothing */
#define XPT SCSI IO 0x01 /* Execute the requested SCSI IO */
#define XPT_GDEV_TYPE 0x02 /* Get the device type information */
#define XPT_PATH_INQ 0x03 /* Path Inquiry */
#define XPT_REL_SIMQ 0x04 /* Release the SIM queue that is frozen */
#define XPT_SASYNC_CB 0x05 /* Set Async callback parameters */
#define XPT_SDEV_TYPE 0x06 /* Set the device type information */
/* XPT SCSI control functions, 0x10 - 0x1F */

```

```

#define XPT_ABORT 0x10 /* Abort the selected CCB */
#define XPT_RESET_BUS 0x11 /* Reset the SCSI bus */
#define XPT_RESET_DEV 0x12 /* Reset the SCSI devices, BDR */
#define XPT_TERM_IO 0x13 /* Terminate the I/O process */

/* Target mode commands, 0x30 - 0x3F */
#define XPT_EN_LUN 0x30 /* Enable LUN, Target mode support */
#define XPT_TARGET_IO 0x31 /* Execute the target IO request */

#define XPT_FUNC 0x7F /* TEMPLATE */
#define XPT_VUNIQUE 0x80 /* All the rest are vendor unique commands */

/* ----- */
/* General allocation length defines for the CCB structures. */
#define IOCDBLEN 12 /* Space for the CDB bytes/pointer */
#define VUHBA 16 /* Vendor Unique HBA length */
#define SIM_ID 16 /* ASCII string len for SIM ID */
#define HBA_ID 16 /* ASCII string len for HBA ID */
#define SIM_PRIV 50 /* Length of SIM private data area */

/* Structure definitions for the CAM control blocks, CCB's for the
  subsystem. */
/* Common CCB header definition. */
typedef struct ccb_header
{
    struct ccb_header *my_addr; /* The address of this CCB */
    u_short cam_ccb_len; /* Length of the entire CCB */
    u_char cam_func_code; /* XPT function code */
    u_char cam_status; /* Returned CAM subsystem status */
    u_char cam_path_id; /* Path ID for the request */
    u_char cam_target_id; /* Target device ID */
    u_char cam_target_lun; /* Target LUN number */
    u_long cam_flags; /* Flags for operation of the subsystem */
} CCB_HEADER;

/* Common SCSI functions. */
/* Union definition for the CDB space in the SCSI I/O request CCB */
typedef struct
{
    u_char *cam_cdb_ptr; /* Pointer to the CDB bytes to send */
    u_char cam_cdb_bytes[ IOCDBLEN ]; /* Area for the CDB to send */
} CDB_UN;

/* Get device type CCB */
typedef struct
{
    CCB_HEADER cam_ch; /* Header information fields */
    u_char cam_pd_type; /* Periph device type from the TLUN */
    char *cam_inq_data; /* Ptr to the inquiry data space */
} CCB_GETDEV;

/* Path inquiry CCB */
typedef struct

```

```

{
  CCB HEADER cam_ch; /* Header information fields */
  u_long cam_feature_flags; /* Supported features flags field */
  u_char cam_version_num; /* Version number for the SIM/HBA */
  u_char cam_hba_inquiry; /* Mimic of INQ byte 7 for the HBA */
  u_char cam_target_sprt; /* Flags for target mode support */
  u_char cam_hba_misc; /* Misc HBA feature flags */
  u_char cam_vuhba_flags[ VUHBA ]; /* Vendor unique capabilities */
  u_long cam_sim_priv; /* Size of SIM private data area */
  u_long cam_async_flags; /* Event cap. for Async Callback */
  u_char cam_hpath_id; /* Highest path ID in the subsystem */
  u_char cam_initiator_id; /* ID of the HBA on the SCSI bus */
  char cam_sim_vid[ SIM_ID ]; /* Vendor ID of the SIM */
  char cam_hba_vid[ HBA_ID ]; /* Vendor ID of the HBA */
  u_char *cam_osd_usage; /* Ptr for the OSD specific area */
} CCB_PATHINQ;

/* Release SIM Queue CCB */
typedef struct
{
  CCB HEADER cam_ch; /* Header information fields */
} CCB_RELSIM;

/* SCSI I/O Request CCB */
typedef struct
{
  CCB HEADER cam_ch; /* Header information fields */
  u_char *cam_pdrv_ptr; /* Ptr used by the Peripheral driver */
  CCB HEADER *cam_next_ccb; /* Ptr to the next CCB for action */
  void (*cam_cbfnp)(); /* Callback on completion function */
  u_char *cam_data_ptr; /* Pointer to the data buf/SG list */
  u_long cam_xfer_len; /* Data xfer length */
  u_char *cam_sense_ptr; /* Pointer to the sense data buffer */
  u_short cam_sense_len; /* Num of bytes in the Autosense buf */
  u_char cam_cdb_len; /* Number of bytes for the CDB */
  u_short cam_sglist_cnt; /* Num of scatter gather list entries */
  u_char cam_scsi_status; /* Returned scsi device status */
  long cam_resid; /* Transfer residual length: 2's comp */
  CDB_UN cam_cdb_io; /* Union for CDB bytes/pointer */
  u_long cam_timeout; /* Timeout value */
  u_char *cam_msg_ptr; /* Pointer to the message buffer */
  u_short cam_msgb_len; /* Num of bytes in the message buf */
  u_short cam_vu_flags; /* Vendor unique flags */
  u_char cam_tag_action; /* What to do for tag queuing */
  u_char cam_sim_priv[ SIM_PRIV ]; /* SIM private data area */
} CCB_SCSIIO;

/* Set Async Callback CCB */
typedef struct
{
  CCB HEADER cam_ch; /* Header information fields */
  u_long cam_async_flags; /* Event enables for Callback resp */
  void (*cam_async_func)(); /* Async Callback function address */
  u_char *pdrv_buf; /* Buffer set aside by the Per. drv */
  u_char pdrv_buf_len; /* The size of the buffer */
} CCB_SETASYNC;

```

```

/* Set device type CCB */
typedef struct
{
  CCB HEADER cam_ch; /* Header information fields */
  u_char cam_dev_type; /* Val for the dev type field in EDT */
} CCB_SETDEV;

/* SCSI Control Functions. */

/* Abort XPT Request CCB */
typedef struct
{
  CCB HEADER cam_ch; /* Header information fields */
  CCB HEADER *cam_abort_ch; /* Pointer to the CCB to abort */
} CCB_ABORT;

/* Reset SCSI Bus CCB */
typedef struct
{
  CCB HEADER cam_ch; /* Header information fields */
} CCB_RESETBUS;

/* Reset SCSI Device CCB */
typedef struct
{
  CCB HEADER cam_ch; /* Header information fields */
} CCB_RESETDEV;

/* Terminate I/O Process Request CCB */
typedef struct
{
  CCB HEADER cam_ch; /* Header information fields */
  CCB HEADER *cam_termio_ch; /* Pointer to the CCB to terminate */
} CCB_TERMIO;

/* Target mode structures. */

typedef struct
{
  CCB HEADER cam_ch;
  u_short cam_grp6_len; /* Group 6 VU CDB length */
  u_short cam_grp7_len; /* Group 7 VU CDB length */
  u_short cam_ccb_listcnt; /* Count of Target CCBs in the list */
  u_char *cam_ccb_listptr; /* Pointer to the target CCB list */
} CCB_EN_LUN;

/* ----- */

/* Defines for the CAM status field in the CCB header. */

#define CAM_REQ_INPROG 0x00 /* CCB request is in progress */
#define CAM_REQ_CMP 0x01 /* CCB request completed w/out error */
#define CAM_REQ_ABORTED 0x02 /* CCB request aborted by the host */
#define CAM_UA_ABORT 0x03 /* Unable to Abort CCB request */
#define CAM_REQ_CMP_ERR 0x04 /* CCB request completed with an err */
#define CAM_BUSY 0x05 /* CAM subsystem is busy */
#define CAM_REQ_INVALID 0x06 /* CCB request is invalid */

```

```

#define CAM_PATH_INVALID      0x07 /* Path ID supplied is invalid */
#define CAM_DEV_NOT_THERE    0x08 /* SCSI device not installed/there */
#define CAM_UA_TERMTO       0x09 /* Unabel to Terminate I/O CCB req */
#define CAM_SEL_TIMEOUT     0x0A /* Target selection timeout */
#define CAM_CMD_TIMEOUT     0x0B /* Command timeout */
#define CAM_MSG_REJECT_REC  0x0D /* Message reject received */
#define CAM_SCST_BUS_RESET  0x0E /* SCSI bus reset sent/received */
#define CAM_UNCOR_PARITY    0x0F /* Uncorrectable parity error ocured */
#define CAM_AUTONSENSE_FAIL 0x10 /* Autosense: Request sense cmd fail */
#define CAM_NO_HBA         0x11 /* No HBA detected Error */
#define CAM_DATA_RUN_ERR   0x12 /* Data overrun/underrun error */
#define CAM_UNEXP_BUSFREE  0x13 /* Unexpected BUS free */
#define CAM_SEQUENCE_FAIL  0x14 /* Target bus phase sequence failure */
#define CAM_CCB_LEN_ERR    0x15 /* CCB length supplied is inadquate */
#define CAM_PROVIDE_FAIL   0x16 /* Unable to provide requ. capability */
#define CAM_BDR_SENT       0x17 /* A SCSI BDR msg was sent to target */
#define CAM_REQ_TERMIO     0x18 /* CCB request terminated by the host */

#define CAM_LUN_INVALID     0x38 /* LUN supplied is invalid */
#define CAM_TID_INVALID    0x39 /* Target ID supplied is invalid */
#define CAM_FUNC_NOTAVAIL  0x3A /* The requ. func is not available */
#define CAM_NO_NEXUS       0x3B /* Nexus is not established */
#define CAM_IID_INVALID    0x3C /* The initiator ID is invalid */
#define CAM_CDB_RECVD      0x3E /* The SCSI CDB has been received */
#define CAM_SCST_BUSY      0x3F /* SCSI bus busy */

#define CAM_SIM_QFRZN      0x40 /* The SIM queue is frozen w/this err */
#define CAM_AUTOSNS_VALID  0x80 /* Autosense data valid for targit */

/* ----- */
/* Defines for the CAM flags field in the CCB header. */

#define CAM_DIR_RESV       0x00000000 /* Data direction (00: reserved) */
#define CAM_DIR_IN        0x00000040 /* Data direction (01: DATA IN) */
#define CAM_DIR_OUT       0x00000080 /* Data direction (10: DATA OUT) */
#define CAM_DIR_NONE      0x000000C0 /* Data directino (11: no data) */
#define CAM_DIS_AUTONSENSE 0x00000020 /* Disable auto sence feature */
#define CAM_SCATTER_VALID 0x00000010 /* Scatter/gather list is valie */
#define CAM_DIS_CALLBACK  0x00000008 /* Disable callback feature */
#define CAM_CDB_LINKED    0x00000004 /* The CCB contains a linked CDB */
#define CAM_QUEUE_ENABLE  0x00000002 /* SIM queue actions are enabled */
#define CAM_CDB_POINTER   0x00000001 /* The CDB field contains a pointer */

#define CAM_DIS_DISCONNECT 0x00008000 /* Disable disconnect */
#define CAM_INITIATE_SYNC  0x00004000 /* Attempt Sync data xfer, and SDTR */
#define CAM_DIS_SYNC       0x00002000 /* Disable sync, go to async */
#define CAM_SIM_QHEAD      0x00001000 /* Place CCB at the head of SIM Q */
#define CAM_SIM_QFREEZE    0x00000800 /* Return the SIM Q to frozen state */

#define CAM_CDB_PHYS       0x00400000 /* CDB pointer is physical */
#define CAM_DATA_PHYS      0x00200000 /* SG/Buffer data ptrs are physical */
#define CAM_SNS_BUF_PHYS   0x00100000 /* Autosense data ptr is physical */
#define CAM_MSG_BUF_PHYS   0x00080000 /* Message buffer ptr is physical */
#define CAM_NXT_CCB_PHYS   0x00040000 /* Next CCB pointer is physical */
#define CAM_CALLBACK_PHYS  0x00020000 /* Callback func ptr is physical */

```

```

#define CAM_DATAB_VALID     0x80000000 /* Data buffer valid */
#define CAM_STATUS_VALID   0x40000000 /* Status buffer valid */
#define CAM_MSGB_VALID     0x20000000 /* Message buffer valid */
#define CAM_TGT_PHASE_MODE 0x08000000 /* The SIM will run in phase mode */
#define CAM_TGT_CCB_AVAIL  0x04000000 /* Target CCB available */
#define CAM_DIS_AUTODISC   0x02000000 /* Disable autodisconnect */
#define CAM_DIS_AUTOSRP    0x01000000 /* Disable autosave/restore ptrs */

/* ----- */
/* Defines for the SIM/HBA queue actions. These value are used in the
SCSI I/O CCB, for the queue action field. [These values should match the
defines from some other include file for the SCSI message phases. We may
not need these definitions here. ] */

#define CAM_SIMPLE_QTAG     0x20 /* Tag for a simple queue */
#define CAM_HEAD_QTAG      0x21 /* Tag for head of queue */
#define CAM_ORDERED_QTAG   0x22 /* Tag for ordered queue */

/* ----- */
/* Defines for the timeout field in the SCSI I/O CCB. At this time a value
of 0xF-F indicates a infinite timeout. A value of 0x0-0 indicates that the
SIM's default timeout can take effect. */

#define CAM_TIME_DEFAULT   0x00000000 /* Use SIM default value */
#define CAM_TIME_INFINITY  0xFFFFFFF /* Infinite timout for I/O */

/* ----- */
/* Defines for the Path Inquiry CCB fields. */

#define CAM_VERSION        0x22 /* Binary value for the current ver */

#define PI_MDP_ABL        0x80 /* Supports MDP message */
#define PI_WIDE_32        0x40 /* Supports 32 bit wide SCSI */
#define PI_WIDE_16        0x20 /* Supports 16 bit wide SCSI */
#define PI_SDTR_ABL       0x10 /* Supports SDTR message */
#define PI_LINKED_CDB     0x08 /* Supports linked CDBs */
#define PI_TAG_ABL        0x02 /* Supports tag queue message */
#define PI_SOFT_RST       0x01 /* Supports soft reset */

#define PIT_PROCESSOR     0x80 /* Target mode processor mode */
#define PIT_PHASE         0x40 /* Target mode phase cog. mode */

#define PIM_SCANHILO      0x80 /* Bus scans from ID 7 to ID 0 */
#define PIM_NOREMOVE      0x40 /* Removable dev not included in scan */

/* ----- */
/* Defines for Asynchronous Callback CCB fields. */

#define AC_FOUND_DEVICES   0x80 /* During a rescan new devies found */
#define AC_SIM_DEREGISTER  0x40 /* A loaded SIM has de-registered */
#define AC_SIM_REGISTER    0x20 /* A loaded SIM has registered */
#define AC_SENT_BDR        0x10 /* A BDR message was sent to target */
#define AC_SCSTI_AEN       0x08 /* A SCSI AEN has been received */

```

```

#define AC_UNSOL_RESEL      0x02 /* A unsolicited reselection occurred */
#define AC_BUS_RESET      0x01 /* A SCSI bus RESET occurred */
/* ----- */
/* Typedef for a scatter/gather list element. */
typedef struct
{
    u_long cam_sg_address; /* Scatter/Gather address */
    u_long cam_sg_count; /* Scatter/Gather count */
} SG_ELEM;
/* ----- */
/* Unix OSD defines and data structures. */
#define INQLEN 36 /* Inquiry string length to store. */
/* General Union for Kernal Space allocation. Contains all the possible CCB
structures. This union should never be used for manipulating CCB's its only
use is for the allocation and deallocation of raw CCB space. */
typedef union
{
    CCB_SCSIIO csio; /* Please keep this first, for debug/print */
    CCB_GETDEV cgd;
    CCB_PATHINQ cpi;
    CCB_RELSIM crs;
    CCB_SETASYNC csa;
    CCB_SETDEV csd;
    CCB_ABORT cab;
    CCB_RESETBUS crb;
    CCB_RESETDEV crd;
    CCB_TERMIO ctio;
} CCB_SIZE_UNION;
/* The typedef for the Async callback information. This structure is used to
store the supplied info from the Set Async Callback CCB, in the EDT table. */
typedef struct
{
    u_short cam_event_enable; /* Event enables for Callback resp */
    void (*cam_async_func)(); /* Async Callback function address */
    u_long cam_async_blen; /* Length of "information" buffer */
    u_char *cam_async_ptr; /* Address for the "information" */
} ASYNC_INFO;
/* The CAM SIM ENTRY definition is used to define the entry points for
the SIMs contained in the SCSI CAM subsystem. Each SIM file will
contain a declaration for it's entry. The address for this entry will
be stored in the cam_conftbl[] array along with all the other SIM
entries. */
typedef struct
{

```

95

```

void (*sim_init)(); /* Pointer to the SIM init routine */
void (*sim_action)(); /* Pointer to the SIM CCB go routine */
} CAM_SIM_ENTRY;
/* The CAM EDT table, this structure contains the device information for
all the devices, SCSI ID and LUN, for all the SCSI busses in the system. */
typedef struct
{
    long cam_tlun_found; /* Flag for the existence of the target/LUN */
    ASYNC_INFO cam_ainfo; /* Async callback info for this B/T/L */
    u_long cam_owner_tag; /* Tag for the peripheral driver's ownership */
    char cam_inq_data[ INQLEN ]; /* storage for the inquiry data */
} CAM_EDT_ENTRY;
/* ----- */

```

Annex D: Operating System Vendor Documentation

D.1 Documentation

The following lists those operating system vendors which have agreed to supply information to third party vendors on the support of SCSI devices. This is not a complete list of all vendors that support CAM.

[ This is a partial list and as it is the addresses of the CAM members it is unlikely to be the right address to contact for information and documentation. Would those listed please advise correct addresses and department and advise the phone number and fax number? Also, any that I should have included, please add yourselves to the list by sending Dal a fax at 408-867-2115. ]

AT&T BELL LABS  
1100 E Warrentonville Rd  
Naperville  
IL 60566

MICROSOFT  
POB 97017  
Redmond  
WA 98073

INTERACTIVE SYSTEMS  
2401 Colorado Ave  
Santa Monica  
CA 90404

NOVELL  
122 E 1700 S  
Provo  
UT 84606

DIGITAL EQUIPMENT  
110 Spit Brook Rd  
ZK03-3/T79  
Nashua  
NH 03062

SUN MICROSYSTEMS  
2550 Garcia Ave Bdg 15  
Mountain View  
CA 94043

IBM  
MS 5226  
PO Box 1328

Boca Raton  
FL 33429

## D.2 DOS Background

During the development of XPT/SIM by the CAM Committee, several approaches to support SCSI under DOS were implemented by vendors. Some were proprietary and required licensing agreements while others were parochial. Of the latter, some provided documentation upon request in order to encourage their adoption.

Subsequent to the XPT/SIM being defined across multiple operating systems, IBM made information available on the attachment of SCSI devices using the 4Bh/80h interrupt, which assumes that there is an SCB (Subsystem Control Block) data structure. SCBs are unlike CCBs.

The IBM introduction created a de facto implementation, and IBM has provided a software license at no charge to those companies which write peripheral drivers to support SCBs. The SCBs and their method of operation are defined in Technical Reference Manuals which are available from IBM.

The CAM approach is oriented towards using the CCB data structure to provide a common approach across multiple operating systems. Although it may have been possible to use SCBs to provide this capability, full technical information was not provided early enough in the development cycle.