

INTERNATIONAL STANDARD

ISO/IEC 9316-2

First edition
2000-06

Information technology – Small computer system interface-2 (SCSI-2) –

Part 2: Common Access Method (CAM) Transport and SCSI interface module



Reference number
ISO/IEC 9316-2:2000(E)

Copyright International Organization for Standardization

INTERNATIONAL STANDARD

ISO/IEC 9316-2

First edition
2000-06

Information technology – Small computer system interface-2 (SCSI-2) –

Part 2: Common Access Method (CAM) Transport and SCSI interface module

© ISO/IEC 2000

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from the publisher.

ISO/IEC Copyright Office • Case postale 56 • CH-1211 Genève 20 • Switzerland



PRICE CODE **XA**

For price, see current catalogue

CONTENTS

	Page
FOREWORD	6
INTRODUCTION	7
Clause	
1 Scope	9
2 Conformance	9
3 Normative references	10
4 Definitions	10
5 Conventions	11
6 General description	12
6.1 General	12
6.2 Environment	12
6.3 Peripheral driver functions	13
6.4 XPT functions	14
6.5 SIM functions	14
7 Background	14
7.1 General	14
7.2 Software	14
7.3 CAM (Common Access Method)	14
7.4 OSD (Operating System Dependencies)	15
7.5 Architectural considerations	15
8 Principles of operation	16
8.1 Accessing the XPT	16
8.2 Initialization	17
8.3 CCB completion	17
8.3.1 Completion of immediate CCB	17
8.3.2 Completion of queued CCB	17
8.4 SCSI request queues	18
8.4.1 The logical unit and the peripheral driver	18
8.4.2 SIM queuing	18
8.5 SIM handling of SCSI resets	20
8.6 Asynchronous event callback	21
8.7 Autosense	23
8.8 Loadable modules	24
9 OSD (operating system dependent) operation	25
9.1 UNIVOS operating system	25
9.1.1 Initialization	25
9.1.2 Accessing the XPT	26
9.1.3 Callback on completion	26
9.1.4 Pointer definition in the UNIVOS environment	26
9.1.5 Request mapping information	27
9.1.6 XPT interface	27
9.1.7 SIM interface	28
9.2 LANOS	28
9.2.1 Initialization	28
9.2.2 SIM and peripheral driver unloading	29
9.2.3 Accessing the XPT	30
9.2.4 Hardware registration	30
9.2.5 Miscellaneous	30

Clause	Page
9.3 DOS (disk operating system)	31
9.3.1 Initialization	31
9.3.2 Accessing the XPT	32
9.3.3 Callback on completion	33
9.3.4 Asynchronous event callbacks	33
9.3.5 Pointer definition	34
10 CAM control blocks	34
10.1 CCB header fields	36
10.1.1 Address of this CCB	36
10.1.2 CAM control block length	36
10.1.3 XPT function code	36
10.1.4 CAM status	36
10.1.5 Connect ID	39
10.1.6 CAM flags	39
10.2 Function codes	39
10.2.1 NOP	39
10.2.2 Get device type	40
10.2.3 Path inquiry	41
10.2.4 Release SIM queue	43
10.2.5 Scan SCSI bus	44
10.2.6 Scan logical unit	45
10.2.7 Set asynchronous callback	45
10.2.8 Set device type	46
10.3 SCSI control functions	47
10.3.1 Abort SCSI command	47
10.3.2 Reset SCSI bus	48
10.3.3 Reset SCSI device	49
10.3.4 Terminate I/O process	49
11 CAM control block to request I/O	51
11.1 Field descriptions for CAM control blocks to request I/O	51
11.1.1 Autosense Residual Length	51
11.1.2 Callback on completion	51
11.1.3 CAM flags	51
11.1.4 CDB	55
11.1.5 CDB length	55
11.1.6 Data transfer length	55
11.1.7 Function code	55
11.1.8 Initiator ID	55
11.1.9 Message buffer length (target-only)	55
11.1.10 Message buffer pointer (target-only)	55
11.1.11 Next CCB pointer	56
11.1.12 Number of scatter/gather entries	56
11.1.13 Peripheral driver pointer	56
11.1.14 Private data	56
11.1.15 Request mapping information	56
11.1.16 Residual length	56
11.1.17 SCSI status	56
11.1.18 Sense info buffer length	56

Clause	Page
11.1.19 Sense info buffer pointer	56
11.1.20 SG list/data buffer pointer	56
11.1.21 Tag ID	56
11.1.22 Tagged queue action.....	57
11.1.23 Timeout value	57
11.1.24 VU field	57
11.2 Execute SCSI I/O	57
11.3 Command linking (optional)	59
11.4 Target mode overview	61
11.5 Phase-Cognizant Mode	62
11.5.1 Enable LUN for phase cognizant mode.....	62
11.5.2 I/O process creation for phase cognizant mode	65
11.5.3 Continuation and completion of an I/O process for phase cognizant mode	66
11.5.4 Non-transparent event handling for phase cognizant mode.....	67
11.5.5 EXECUTE TARGET I/O CCB.....	69
11.6 Host Target Mode.....	71
11.6.1 Host Target Mode functionality not specified	71
11.6.2 Host Target Mode messages	71
11.6.3 Use of the IMMEDIATE NOTIFY CCB.....	72
11.6.4 IMMEDIATE NOTIFY CCB.....	79
11.6.5 NOTIFY ACKNOWLEDGE CCB	81
11.6.6 Enable target mode LUN for Host Target Mode	81
11.6.7 ENABLE LUN CCB for Host Target Mode	84
11.6.8 ACCEPT TARGET I/O and CONTINUE TARGET I/O CCB operation	85
11.6.9 ACCEPT TARGET I/O CCB	91
11.6.10 CONTINUE TARGET I/O CCB	92
11.6.11 Disable of a Host Target Mode LUN	94
11.6.12 Exception conditions	94
11.6.13 CDB reception on a non enabled LUN	96
11.6.14 Retrieving unused ACCEPT TARGET I/O CCBs from the SIM	96
12 HBA engines	96
13 Engine inquiry	97
13.1 Execute engine (optional).....	98
Annex A (informative) Physical/logical translation in 80x86 environment.....	100
Annex B (informative) Target peripheral driver example	105
Annex C (informative) UNIVOS OSD data structures.....	107
Figure 1 – CAM environment model	13
Figure 2 – CAM layers	29

	Page
Table 1 – Asynchronous event callback opcode data requirements	23
Table 2 – CAM control block header	34
Table 3 – Support of SCSI messages.....	35
Table 4 – XPT function codes	36
Table 5 – CAM status	37
Table 6 – NOP CCB.....	40
Table 7 – Get device type CCB	40
Table 8 – PATH INQUIRY CCB – Part 1 of 2.....	41
Table 9 – Release SIM queue	44
Table 10 – SCAN SCSI BUS CCB.....	44
Table 11 – SCAN LOGICAL UNIT CCB	45
Table 12 – SET ASYNCHRONOUS CALLBACK CCB.....	46
Table 13 – SET DEVICE TYPE CCB	46
Table 14 – ABORT SCSI COMMAND CCB.....	48
Table 15 – RESET SCSI BUS CCB.....	48
Table 16 – RESET SCSI DEVICE CCB	49
Table 17 – TERMINATE I/O PROCESS CCB	50
Table 18 – CAM flags – Part 1 of 2	52
Table 19 – Scatter Gather List	53
Table 20 – EXECUTE SCSI I/O REQUEST CCB	58
Table 21 – ENABLE LUN CCB for phase cognizant mode	63
Table 22 – CCB List.....	63
Table 23 – EXECUTE TARGET I/O CCB.....	70
Table 24 – IMMEDIATE NOTIFY CCB.....	80
Table 25 – NOTIFY ACKNOWLEDGE CCB	81
Table 26 – ENABLE LUN CCB for Host Target Mode	84
Table 27 – ACCEPT TARGET I/O CCB	91
Table 28 – CONTINUE TARGET I/O CCB	93
Table 29 – ENGINE INQUIRY CCB	97
Table 30 – EXECUTE ENGINE REQUEST CCB.....	98

INFORMATION TECHNOLOGY – SMALL COMPUTER SYSTEM INTERFACE-2 (SCSI-2) –

Part 2: Common Access Method (CAM) Transport and SCSI interface module

FOREWORD

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

International Standard ISO/IEC 9316-2 was prepared by subcommittee 25: Interconnection of information technology equipment, of ISO/IEC joint technical committee 1: Information technology.

Annexes A, B and C are for information only.

INTRODUCTION

SCSI provides 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 have made it nearly impossible for end users 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. That effort has evolved into this International Standard.

– Blank page –

.....

INFORMATION TECHNOLOGY – SMALL COMPUTER SYSTEM INTERFACE-2 (SCSI-2) –

Part 2: Common Access Method (CAM) Transport and SCSI interface module

1 Scope

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

The purpose of this International 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.

2 Conformance

An implementation claiming conformance to the transport layer (XPT) for a specified operating system and language environment shall:

- provide all the mandatory XPT functions and services specified in this International Standard;
- correctly interoperate with any conforming SCSI Interface Module (SIM) for the specified environment;
- provide the necessary interface specifications that a conforming SIM requires to interface with the XPT.

An implementation claiming conformance to the SIM for a specified operating system and language environment shall:

- provide all the mandatory SIM functions and services specified in this International Standard;
- correctly interoperate with any conforming XPT for the specified environment;
- provide the necessary interface specifications that a conforming XPT requires to interface with SIMs.

A conforming implementation shall execute all function codes as required by this International Standard, and in response to these codes shall only return specified status, and return codes. A conforming implementation may provide additional capabilities via Vendor Unique function codes.

If an operating system is not specified in this International Standard, then that operating system shall conform to 9.1 in this International Standard. (See also annex C.)

Claims of conformance to this International Standard shall state:

- whether conformance is claimed with the XPT or the SIM or both;
- which operating systems and environments are supported;
- whether the optional capabilities of target mode or Host Bus Adaptor (HBA) engines are supported.

3 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 9316. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 9316 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of IEC and ISO maintain registers of currently valid International Standards.

ISO/IEC 9316:1995, *Information Technology – Small Computer Interface-2*

4 Definitions

For the purposes of this International Standard, the following definitions apply:

4.1

block

defines an action to prevent access, (e.g., to obstruct the action of or the continuation of a process thread)

4.2

(CAM control block) CCB

data structure provided by peripheral drivers to the XPT to control execution of a function by the SIM

4.3

immediate CCB

provides valid completion status when the call to xpt_action () returns (e.g., path inquiry)

4.4

queued CCB

provides valid completion status when the completion callback routine is called, or the CAM Status field in the CCB changes from Request In Progress to another valid CAM Status

4.5

CDB (command descriptor block)

data structure containing the SCSI opcode, parameters, and control bits for that operation

4.6

DMA (direct memory access)

means of data transfer between peripheral and host memory without processor intervention

4.7

freeze

defines a software action to quiesce activity (e.g., freeze the queue)

4.8

HBA (host bus adapter)

hardware and microcode which provides the interface between system memory and any number of SCSI buses

4.9

null

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

4.10**optional**

this term describes features which are not required by this International Standard. However, if any feature defined by this International Standard is implemented, it shall be done in the same way as defined by the International 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

4.11**OSD (Operating System Dependent)**

this term describes a capability, method of operation, or feature that depends on the specific operating system on which CAM is implemented

4.12**path**

this term describes the address of the XPT or a SIM/HBA SCSI bus combination

4.13**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

4.14**SIM (SCSI interface module)**

module designed to accept the CAM control blocks routed through the XPT in order to execute SCSI commands and perform other functions

4.15**VU (vendor unique)**

this term is used to describe bits, bytes, fields, code values, and features which are not described in this International Standard, and may be used in a way that varies among vendors

4.16**XPT (transport)**

a layer of software which peripheral drivers use to originate the execution of CAM functions

5 Conventions

Within the tables, there is a direction bit which indicates in or out. The direction is from the view point 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, commands and messages. 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 English meaning).

There are places in this International Standard where programming language is used to define or express a concept in order to assist the reader. These are not copyrighted program steps and implementors are encouraged to use the code wherever it suits their application.

6 General description

6.1 General

The application environment for CAM is any computer addressing a SCSI peripheral through a protocol chip on a motherboard or a host bus adapter.

SCSI is a widely-used interface which provides common attachment for a variety of peripherals.

The purpose of the CAM is to define a International Standard for the support of Host Bus Adapters (HBA) and the like by device driver software.

Software in the operating system dispatches I/O requests to the SCSI peripherals in a number of different ways depending on the software architecture. The operating system dependencies (OSD) are defined in 9 for certain named software and hardware platforms.

6.2 Environment

A model of the CAM usage environment is illustrated in figure 1. Multiple applications are shown accessing a variety of SCSI devices. Several device drivers, both peripheral drivers and SIMs, are present to support the peripherals on the system.

The choice of XPT and SIM packaging is an operating system dependency. Clause 9 defines this dependency for certain named software and hardware platforms.

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

The XPT function is illustrated as a separate element. In some operating systems, the XPT operations are incorporated into a single module which integrates both XPT and SIM functionality. The logical separation between the two is maintained because there may be more than one SIM loaded. The XPT function may be provided by the operating system, or can be achieved through associating multiple separately loaded XPTs.

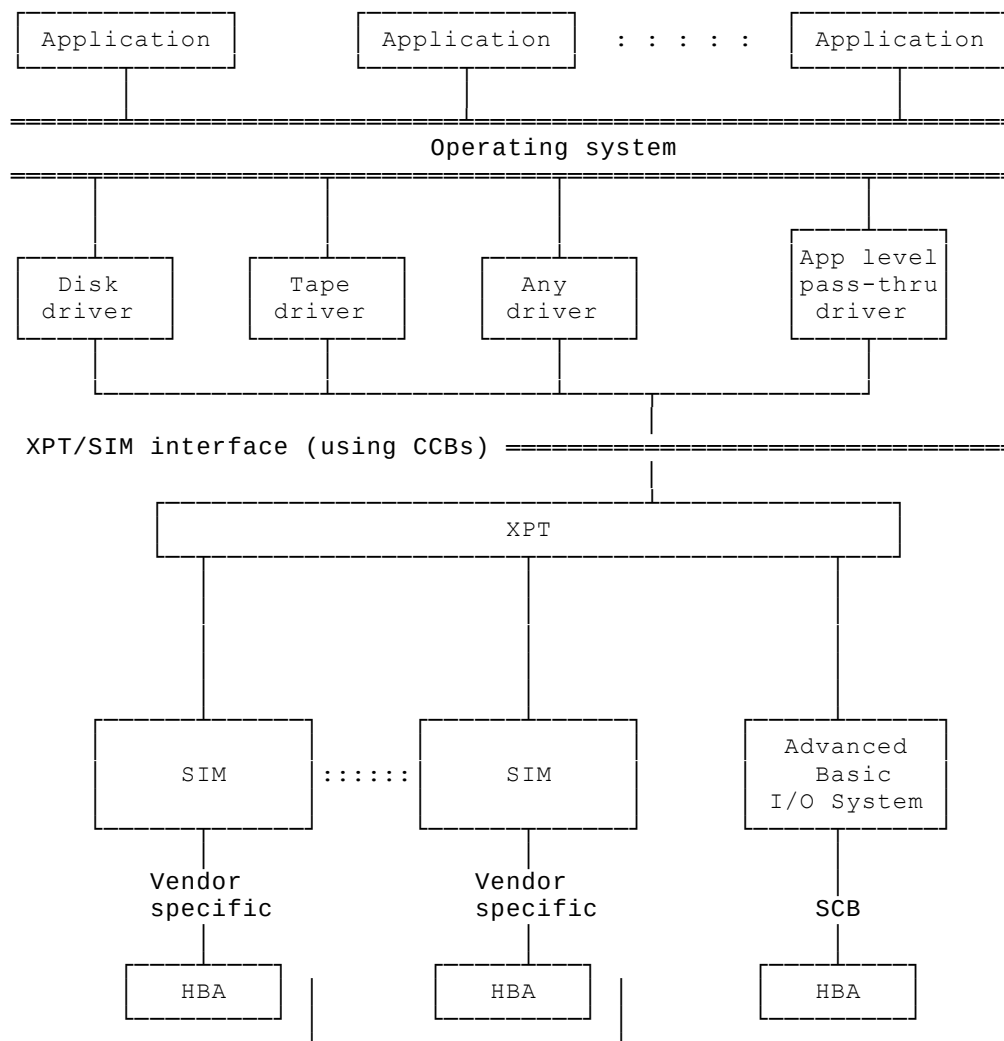


Figure 1 – CAM environment model

6.3 Peripheral driver functions

Peripheral drivers provide the following functionality:

- a) interpreting 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
 - 1) CAM control blocks and supporting blocks that may be needed,
 - 2) 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) establishing 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 utility at installation and formatting time;
- j) establishing a timeout value for a task and passing this value in the CCB.

6.4 XPT functions

XPT services provide the following functionality to process CCBs:

- a) routing of the CCB to the proper SIM;
- b) OSD management 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 drivers.

6.5 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 appropriate) all the SCSI HBA protocol steps;
- c) distinguish abnormal behavior and perform error recovery, as required;
- d) manage data transfer path hardware, including DMA circuitry and address mapping, and establish DMA resource requests (if necessary);
- e) queuing of multiple operations for different LUNs as well as the same LUN and assign tags for tag queuing (if supported);
- f) freeze and unfreeze the queue as necessary to accomplish queue recovery;
- g) post the completed operation to the initiating device driver;
- h) manage selection, disconnection, reconnection, and data pointers of the SCSI HBA protocol;
- i) implement a timer mechanism, using values provided by the peripheral driver.

7 Background

7.1 General

SCSI 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.

7.2 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.

7.3 CAM (Common Access Method)

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

- 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 responsibilities of the transport.

- SIM (SCSI Interface Module):

the SIM (SCSI interface module) manages HBA resources and provides a hardware-independent interface for SCSI applications and drivers. 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.

- 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.

7.4 OSD (Operating System Dependencies)

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

Although the fields of a CCB have a common meaning, the contents may 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.

Some OSD issues are discussed in clause 9.

7.5 Architectural considerations

Ideally, a single XPT structure would suffice for all OS environments, 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 ascertains 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 International Standard.

Under a UNIVOS (Universal Operating System), the XPT and SIMs 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 system if suitable routing facilities are provided by the system vendor. One example of a universal operating system is UNIX (a registered trademark of X/Open Ltd.) and as described in this International Standard, software developed for a UNIVOS could be compatible within a Unix system.

Under a LANOS (Local Area Networking Operating System), the XPT can be loaded as a loadable device driver. SIMs are also implemented as loadable device drivers. One example of a local area network operating system is Netware 386 (a Trademark of Novell) and as described in this International Standard, software developed for a LANOS could be compatible within a Netware 386 system.

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

XPT routing is a mechanism to support concurrent multiple co-resident SIMs so that different HBAs can be present in the same system. This task is handled by the XPT logical entity. The XPT is implemented differently under each operating system, but the logical functionality is the same for all operating systems.

Once one or more SIMs are loaded, the peripheral drivers integrate each type of SCSI device into the OS through the XPT, independent of the installed HBA hardware.

This is a list of requirements that CAM is designed to meet:

- a) the ability to use any SCSI command (including vendor unique);
- b) no restrictions on the size or format of transferred data;
- c) allow all the capabilities of high end host adapters to be fully utilized and accommodating HBAs which do most of the SCSI processing on board (this precludes interfaces which expect to control SCSI phases);
- d) allow the calling peripheral driver or program to interpret sense data returned by SCSI devices;
- e) fully re-entrant code;

NOTE This is an obvious requirement for multi-tasking environments such as UNIVOS but even in single tasking DOS applications, multi-threaded 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 multi-threading, requests can be queued, and serialized within the SIM module transparently to the XPT.

- f) support of multiple HBAs;
- g) peripheral drivers may ascertain which optional features are available;
- h) define a single XPT-based SCSI Bus scanning algorithm (so that each peripheral driver need not use bus bandwidth to perform this function);
- i) provide a mechanism to abort I/O requests (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.

8 Principles of operation

8.1 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 responsible for routing a CCB to the SIM indicated by the Path ID field.

The XPT is not involved in the reverse process of advising the peripheral driver of the completion of a queued 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). Completion notification for immediate requests is accomplished by the normal function call return mechanism.

The XPT is responsible for notifying peripheral drivers of asynchronous events via the asynchronous callback mechanism.

8.2 Initialization

The XPT is responsible for determining the SCSI Bus configuration at initialization for each SIM.

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

```

for all SCSI buses
    for all SCSI IDs (excluding the HBA SCSI ID)
        find the device
            if device exists
                for all LUNs
                    send an INQUIRY command and save returned
                    information
                end for
            end if
        end for
    end for
end for

```

8.3 CCB completion

CCB completion is either immediate or queued.

8.3.1 Completion of immediate CCB

All CCBs are complete when the xpt_action() call returns except:

- Execute SCSI I/O;
- Execute Target I/O;
- Accept Target I/O;
- Continue Target I/O;
- Immediate Notify;
- Execute Engine Request.

8.3.2 Completion of queued CCB

The XPT/SIM shall set all appropriate queued CCB fields marked as IN or IN/OUT and shall provide autosense information as specified in this International Standard before CCB completion notification is done. See 8.7 and the appropriate CCB tables in clauses 10, 11 and 12 for further information.

A peripheral driver is notified of the completion of a queued CCB by using one of the following mechanisms, as directed by the CAM Flags field in the CCB.

- CCB Callback on completion.

The SIM/XPT calls the peripheral driver's callback routine when execution of the queued CCB completes. The queued CCB includes a pointer to the peripheral driver's callback routine (in the Callback on Completion field).

The peripheral driver callback routine is used much like a hardware interrupt handler. The callback routine has the same privileges and restrictions as an interrupt handler.

The address of the specific CCB that completed is passed to the peripheral driver's callback routine.

- CCB CAM Status field change from Request in Progress to another CAM Status.

If the CCB CAM Flags has the Disable Callback of Completion set the XPT/SIM shall notify the peripheral driver of a completed CCB by changing the CAM Status field of the CCB from Request in Progress to another valid CAM Status. The peripheral driver shall be responsible for monitoring the CCB CAM Status field for completion.

8.4 SCSI request queues

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

A SCSI request can be queued in any of the following places:

- a) in the SIM;
- b) in the target/LUN;
- c) in the peripheral driver.

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

SCSI-2 permits a Logical Unit to keep a queue using tag queues, or a simple queue of one element.

A peripheral driver may also keep a queue (e.g., to perform a simple elevator sort).

8.4.1 The logical unit and the peripheral driver

Based on needs outside the scope of CAM, a peripheral driver may maintain a list of unfinished queued CCBs. The SIM, acting on behalf of the peripheral driver, sends the appropriate commands or messages to manage the queues in the Logical Unit. When the Logical Unit completes an operation, the peripheral driver is advised by the SIM via a callback or by checking CAM Status for completion (see 8.3 for additional information).

The peripheral driver should be coded as if there are other peripheral drivers and other initiators working with the same Logical Unit.

8.4.2 SIM queuing

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

8.4.2.1 SIM queue priority

When a CCB has a SIM queue priority=1, the SIM places the CCB ahead of all CCBs that have a SIM queue priority=0 for the Logical Unit and at the end (tail) of all other CCBs having a SIM queue priority=1. 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 of CCBs that have SIM priority=0 and the queue remains frozen. When the SIM

queue is released, any CCBs with SIM queue priority=1 are executed individually first (in FIFO sequence), followed by CCBs with a SIM priority=0 (in FIFO sequence). See 8.4.2.3 on how to release the SIM queue.

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.

8.4.2.2 Tag recognition

To support tagged queuing recognition the SIM maintains a reference between the CCB pointers and the queue tags for a Logical Unit. By this means, the SIM can handle both the queue tag resource allocation and reconnection of the I_T_L_Q nexus (see ISO/IEC 9316) for further information on I_T_L_Q nexus) for the CCB from a peripheral driver.

The peripheral driver shall allow the SIM/XPT to handle the assignment of the queue tag ID for the request. The SIM shall assign unique TAG Ids for the Logical Unit operation based on its internal reference table.

When a Logical Unit that supports tagged queuing reconnects to the initiator (SIM/HBA pair), it sends 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 continue the SCSI transaction. The queue tag ID shall be freed by the SIM at the completion of the SCSI request.

8.4.2.3 Error conditions and queues within the subsystem

The SIM shall place its internal queue for a Logical Unit into the frozen state for any status other than Request Completed without Error or Request in Progress for an EXECUTE SCSI I/O REQUEST CCB, unless the SIM queue freeze disable bit has been set in the CCB. If the SIM queue freeze disable bit is set, the queue shall not be frozen and CCB execution continues from the SIM queue.

NOTE 1 Immediate CCB function types, Target Mode CCBs, and EXECUTE ENGINE REQUEST CCBs never result in a frozen SIM queue. The action of some immediate CCBs (e.g., Abort SCSI Command, Terminate I/O Process Request) may result in an EXECUTE SCSI I/O REQUEST CCB being returned which does freeze the SIM queue.

The SIM shall maintain a count of the number of CCBs returned with the indication of SIM queue frozen. The SIM shall decrement the SIM queue frozen count for each CCB received with a Release SIM Queue function code with the SIM Queue Freeze bit set to a one. The SIM/HBA shall not release its internal queue until the SIM queue frozen count is zero. The SIM shall not allow the SIM queue frozen count to have a negative value. The SIM shall not consider it an error when a CCB is received with a Release SIM Queue function code that would decrement the SIM queue frozen count past zero (negative).

In the event that a SIM encounters an error condition which can not be associated with a CCB the SIM shall not freeze the queue. The SIM should attempt to continue operation. Failing that the SIM shall take corrective action leaving the SCSI bus in a usable state.

When a SIM's Logical Unit's queue is in the frozen state (a positive value in the SIM queue frozen count), the SIM shall not dispatch any CCBs to that Logical Unit except to retrieve autosense information. Peripheral drivers can still send CCBs to the SIM for the Logical Unit, or any other Logical Unit. Any new CCBs received by the SIM shall be placed in the queue according to the rules specified in 8.4.2.1.

NOTE 2 Since the SIM is the initiator, the SIM's internal queue goes into a frozen state so that the pending sense information in the Logical Unit is not discarded. The SIM holds its internal Logical Unit queue in the frozen state until RELEASE SIM QUEUE CCB(s) are received that decrement the SIM queue frozen count to zero.

NOTE 3 The SIM queue frozen count can be greater than one for a number of conditions. An example of two of these conditions are SCSI BUS RESET, and a target operating in tagged queue mode.

Using the CCB, the SIM returns in the CAM Status field an indication of the frozen queue condition and other information. The peripheral driver acts upon the information returned via the CCB. The setting of the Autosense bit in the CAM Flags field 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 the sense data does not release the SIM queue). See 8.7 for further information on autosense.

8.5 SIM handling of SCSI resets

CAM defines the SIM/HBA mandatory handling of the SCSI hard reset alternative and SCSI BUS DEVICE RESET messages. The optional handling of the SCSI soft reset alternative for SCSI bus resets is SIM/HBA vendor unique. CAM defines some operational functionality that shall be adhered to if the SIM/HBA supports the SCSI bus soft reset alternative.

The SIM/HBA handling of the SCSI hard bus reset shall be as follows.

- Upon detecting a SCSI bus reset, the SIM/HBA shall perform the following actions in sequence:
 - a) block Path ID to the reset bus, all new received CCBs for this bus are rejected with status of CAM busy;
 - b) return all CCBs within the SIM/HBA for this bus with a CAM status of SCSI Bus Reset Sent/Received;
 - c) call: `xpt_async(opcode=reset, path_id=bus that was reset, target_id=-1, lun=-1, buffer_ptr=null, data_cnt=0);`
 - d) unblock Path ID for the bus;
 - e) resume normal processing of CCBs.

For the operational functionality for SCSI soft reset alternative, the SIM/HBA shall implement in a vendor unique manner as follows.

- The SIM/HBA shall check whether all Logical Units for a SCSI bus it controls implement the SCSI soft reset alternative.
- If all Logical Units support the SCSI soft reset alternative for a SCSI bus, the SIM/HBA shall indicate its support for the soft reset alternative by setting the Soft Reset flag in the PATH INQUIRY CCB.
- Upon detecting a SCSI bus reset for a SIM/HBA that has indicated support of the SCSI soft reset alternative in the PATH INQUIRY CCB, the SIM/HBA shall perform the following actions in sequence:
 - a) handle the SCSI soft reset alternative as defined in ISO/IEC 9316;
 - b) call: `xpt_async(opcode=reset, path_id=bus that was reset, target_id=-1, lun=-1, buffer_ptr=null, data_cnt=0);`
 - c) resume normal processing of CCBs.
- If not all Logical Units for a SCSI bus implement the SCSI soft reset alternative, the SIM/HBA shall ensure in a vendor unique manner that all Logical Units operate in consistent manner with either the SCSI soft reset alternate or the hard reset alternative. The SIM/HBA shall also indicate the alternative chosen by the PATH INQUIRY CCB Soft Reset flag setting.

The SIM/HBA handling of the SCSI BUS DEVICE RESET message shall be as follows.

- Upon successfully issuing a SCSI BUS DEVICE RESET message to a target, the SIM/HBA shall perform the following actions in sequence:
 - a) block Target Id for the bus, all new received CCBs for this target are rejected with status of CAM busy;
 - b) return all CCBs within the SIM/HBA for this target with CAM status of Bus Device Reset sent;

- c) call: `xpt_async(opcode=reset, path_id=bus that issued the SCSI BUS DEVICE RESET message, target_id= target id of device that received the SCSI BUS DEVICE RESET message, lun=-1, buffer_ptr=null, data_cnt=0);`
- d) unblock Target ID for the bus;
- e) resume normal processing of CCBs.

8.6 Asynchronous event callback

In an event such as a SCSI bus reset or an asynchronous event notification the XPT/SIM 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. The peripheral driver shall return from the callback.

During system startup or peripheral driver initialization, the peripheral driver should register for an asynchronous event callbacks for each Logical Unit with which it is working and once with the XPT. The Asynchronous Events for Logical Units registration are the following:

- Sent Bus Device Reset to target;
- SCSI AEN;
- Unsolicited reselection;
- SCSI Bus Reset.

The Asynchronous Events for the XPT registration are the following:

- New Logical Units found during a rescan;
- Path ID de-registered;
- Path ID registered.

In order for a peripheral driver to receive asynchronous event callbacks for a Logical Unit, it shall issue a SET ASYNCHRONOUS CALLBACK CCB addressed to the Logical Unit with the Asynchronous Event Enables fields set to a 1 for those events the peripheral driver wishes to be notified of through an asynchronous callback. For the XPT asynchronous event callbacks, the peripheral driver shall issue a SET ASYNCHRONOUS CALLBACK CCB addressed to the XPT (Path ID 0xFF) with the Asynchronous Event Enables fields set to a one for those events the peripheral driver wishes to be notified of through an asynchronous callback. The SET ASYNCHRONOUS CALLBACK CCB shall apply only to a single Logical Unit or the XPT. The use of wildcards shall not be supported for the SET ASYNCHRONOUS CALLBACK CCB.

The peripheral driver can change its asynchronous events callbacks for a particular SCSI Logical Unit or the XPT by issuing the SET ASYNCHRONOUS CALLBACK CCB to a Logical Unit or the XPT, with the Asynchronous Event Enables field set to the replacement value, an updated Peripheral Driver Buffer Pointer field, an updated Size of Allocated Peripheral Buffer field, and the Asynchronous Callback Pointer field containing the original registered value.

The peripheral driver can de-register for asynchronous event callbacks for a Logical Unit or the XPT by issuing the SET ASYNCHRONOUS CALLBACK CCB to the Logical Unit or the XPT with the Asynchronous Event Enables field set to zero and the Asynchronous Callback Pointer field containing the original registered value. When a peripheral driver wishes to change its asynchronous event callback routine, it shall do so by de-registering and then shall follow the registration procedure.

The XPT shall pass all SET ASYNCHRONOUS CALLBACK CCBs to the addressed (Path ID) SIM in addition to performing XPT table maintenance. This allows the SIM to perform its own asynchronous callbacks to peripheral drivers, foregoing the available (and required) XPT services. The callback interface to the peripheral driver is the same whether the asynchronous callback is from the XPT or directly from the SIM.

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

- a) Classify the event
ascertain the opcode. (see Table 1).
- b) Format the associated data within an internal (to the SIM) buffer (e.g., the sense data received from an AEN).
- c) Perform the XPT reverse routing required by the event. The SIM calls the async callback entry point in the XPT:
`long xpt_async(opcode, path_id, target_id, lun, buffer_ptr, data_cnt).`

The XPT shall be responsible for ensuring that requests to the `xpt_async()` routine are processed in serial fashion. Alternatively, the SIM shall call the peripheral driver directly. In this case the SIM shall be responsible for serialization.

All of the arguments, other than the pointer, are long values of 32 bits. The value of –1 in `path_id`, `target_id`, and `lun` fields can be used as a wildcard. A null buffer pointer value and a count of 0 are valid for opcodes that do not require any data transfer.

Using the `path_id`, `target_id`, `lun` fields, and event opcode information available directly to the SIM or to the XPT from the `xpt_async()` call, the XPT or SIM scans its internal tables looking for "matches" with the registered asynchronous callback peripheral drivers (see 10.2.7). When a match is found, either exactly or with a wildcard of –1, the XPT or SIM 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.

```
void (*cam_async_func) (opcode,path_id,target_id,lun,buffer_ptr,data_cnt)
```

The `buffer_ptr` value shall be the peripheral driver's buffer. The `data_cnt` value shall be what the XPT has to transfer from the SIM's buffer up to the limit of the peripheral driver's buffer.

For most opcodes the `path_id`, `target_id`, and `lun` arguments are the only ones needed. The only opcodes that require an additional buffer are AEN, Path ID registered, and Path ID de-registered.

Table 1 lists the opcodes and the expected data requirements for the number of bytes to be transferred.

Table 1 – Asynchronous event 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 reserved	0x0002 0x0004	Valid	Valid	Valid	n/a
SCSI AEN	0x0008	Valid	Valid	Valid	Min. 22
Sent BDR to target	0x0010	Valid	Valid	n/a	n/a
Path ID registered	0x0020	XPT ID	n/a	n/a	Min. 1
Path ID de-registered	0x0040	XPT ID	n/a	n/a	Min. 1
New devices found	0x0080	Valid	n/a	n/a	n/a

The AEN requires 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 ISO/IEC 9316).

The Path ID registered and the Path ID de-registered data requirements shall be a minimum of 1 byte. This byte shall contain the Path ID needed to access the SIM. Since Path ID registered and Path ID de-registered Asynchronous Callback Requests are directed to the XPT, the Path ID argument is the XPT's Path ID (0xFF).

The new devices found opcode shall be returned whenever the XPT/SIM issues an inquiry which detects that a device is attached which was not previously found (e.g., a printer powered on after system initialization was completed).

NOTE Some devices provide minimal information at power-on and cannot provide complete inquiry information until after some delay. A XPT/SIM may scan the bus after initialization to update its tables with the complete inquiry information.

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

8.7 Autosense

Autosense causes sense data to be retrieved automatically if a CHECK CONDITION or COMMAND TERMINATED status is reported in the SCSI Status field of the CCB. Unless the Disable Autosense CAM Flag is a 1, the SIM shall perform an Autosense function as described in this subclause.

The SIM shall construct a REQUEST SENSE command and send it to the same Logical Unit. The location and amount of sense data are specified in the Sense Info Buffer Pointer and Sense Info Buffer Length fields respectively of the SCSI I/O Request CCB. If the length field is 0 or the Sense Info Buffer Pointer field is null, the REQUEST SENSE command shall be issued, but with a data allocation length of 0.

After completing the autosense sense sequence without failure the CAM Status and SCSI Status fields shall contain the status of the original command that caused the check condition.

The target can return fewer than the number of sense bytes requested. This is not reported as an error, sense status shall be flagged as valid and the Autosense Residual Length field shall be set correctly.

8.8 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 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 (Path IDs) that are present in the system and targets that have been identified as being on those HBAs.

When a SIM is loaded, the SIM and XPT have to work together to have the SIM-supported HBAs registered as addressable Path IDs. 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 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;
```

When the xpt_bus_register function is called, the XPT shall 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.

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_bus_deregister function is called, the XPT shall update its internal tables to reflect that the path_id (HBA) is not present.

Peripheral drivers can request to be informed when a Path ID is registered or de-registered via the async callback feature (see 8.6 and 10.2.7).

9 OSD (operating system dependent) operation

9.1 UNIVOS operating system

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

There are several sets of modules for UNIVOS:

- 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 UNIVOS 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 18.

The success of a function is reported with a CAM Status of Request Completed without Error.

The failure of a function is reported with any other CAM Status except Request In Progress.

9.1.1 Initialization

The initialization of the XPT and SIMs is under the control of the configuration_driver.

Due to the different UNIVOS-based systems, 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 Logical Unit 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 ascertain 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 Logical Units 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 return the inquiry data, even though the device may not be ready for other commands. A limited number of retries are 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 is set to "not found". The inquiry command shall be the only command issued by the XPT to the devices during initialization.

9.1.2 Accessing the XPT

9.1.2.1 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.

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 its 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;
- the `xpt_ccb_alloc()` routine shall return a null if memory resources are not immediately available.

9.1.2.2 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 `"long (* 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).

9.1.3 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 `cbfcnp`, and declared as `"void (*cbfcnp)(CCB *)"`. The argument to `cbfcnp` shall be the address to the CCB.

The Disable Callback on Completion feature should not be used. Peripheral drivers should not poll the CAM Status field.

9.1.4 Pointer definition in the UNIVOS environment

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

9.1.5 Request mapping information

This field is expected to contain a pointer to the buffer 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.

9.1.6 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.

9.1.6.1 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 value 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.

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 CAM Status.

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 value is either success or failure.

9.1.6.2 Functions for SIM module support

a) See 8.8 for loadable module support:

long xpt_bus_register(CAM_SIM_ENTRY *)

long xpt_bus_deregister(path_id)

b) void 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 Logical Unit 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.

9.1.7 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(path_id)

This routine is called by the XPT to request that the SIM be initialized. The argument is the Path Id assigned to the SIM and the returned value 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 a CCB, and the returned value is either success or failure.

9.2 LANOS

LANOS drivers are called NLMs (LANOS loadable modules). These modules are registered and linked dynamically with LANOS: they are loaded after the operating system is initialized and may be unloaded at any time.

The LANOS 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 loaded.

The CAM Flags used are as described in Table 18.

9.2.1 Initialization

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

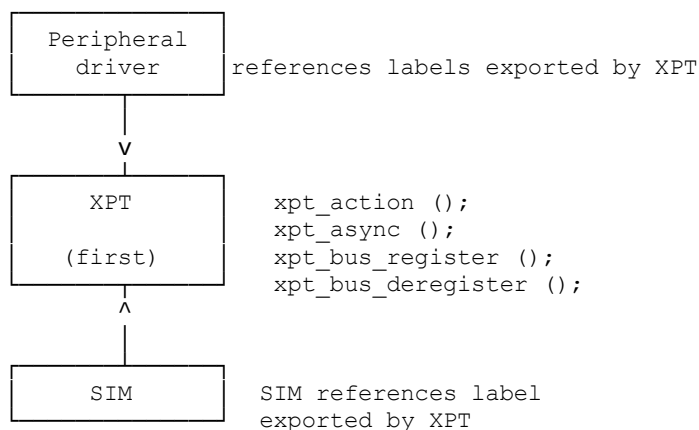


Figure 2 – CAM layers

For an overview of SIM SCSI registration with the XPT see 8.8. For an overview of peripheral driver registration with the XPT see 8.6 and 10.2.7.

When LANOS loads a SIM, it shall call the initialization routine specified in the LANOS 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 supports, 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 LANOS loads a peripheral driver, the initialization routine specified in the linker definition file shall be called. At this time, the driver needs to ascertain which, if any, SIMs are registered.

The peripheral driver sends a PATH INQUIRY CCB to each path to ascertain if a SIM is registered. If a valid response is returned the peripheral driver checks for devices that it supports 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.

9.2.2 SIM and peripheral driver unloading

Before a SIM unloads, 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 LANOS that the drives on this path are in an inactive state. The XPT then removes the path from its internal tables and further peripheral driver requests on this path shall return CAM Status of Invalid Path ID.

Before a peripheral driver unloads, 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 then removes this driver from its callback tables.

The XPT can only be unloaded after all peripheral drivers and SIM modules have been unloaded. LANOS does 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, LANOS does 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 unloaded.

9.2.3 Accessing the XPT

LANOS allows an NLM to export functions which NLMs loaded 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 control 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 by the SIM to register a SCSI bus with the XPT and obtain a Path ID for the SCSI bus;
- `xpt_bus_deregister ()` is used to de-register the passed Path ID and associated SCSI bus.

9.2.4 Hardware registration

The SIM module needs to do the actual registration of the host adapter with LANOS. 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 LANOS, only the hardware options used by the card (e.g., interrupt line, base address, DMA, etc).

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

A peripheral driver registers a logical card with LANOS for each `path_id` it supports. This logical card uses no hardware resources, but does have entry points for I/O and IOCTL requests from LANOS. The peripheral driver also reports the devices that it supports to LANOS.

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

9.2.5 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 supports, so the memory can be allocated in the data structure provided by LANOS when a device is added to the system.

Since fast disk channels are essential for a LANOS 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 calls 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 LANOS routines that are "blocking" or the file server abends.

9.3 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 ascertains 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 18.

9.3.1 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.

9.3.1.1 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 9.3.2.1)
    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;

```

9.3.1.2 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).

9.3.2 Accessing the XPT

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

9.3.2.1 Testing for the presence of the XPT/SIM

Peripheral drivers and applications can check for the presence of an XPT/SIM module (e.g., 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,1           ; 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

```

9.3.2.2 Sending a CCB to the XPT

Once it is ascertained 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 e.g.

On entry:

ES:BX = address of the CCB

AX = 8100h

On return:

AH = 0 if successful

= any other value if an error occurred

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

```

9.3.3 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, all hardware interrupts shall be disabled and ES:BX shall point to the completed CCB.

9.3.4 Asynchronous event callbacks

There are some differences in the DOS XPT/SIM implementation of asynchronous callbacks as compared with the description in 8.6.

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 ASYNCHRONOUS CALLBACK CCB is held by the XPT/SIM until it is "de-registered". This is accomplished by sending another SET ASYNCHRONOUS CALLBACK CCB to the XPT/SIM with all of the Asynchronous Event Enables field reset and the address of the original SET ASYNCHRONOUS 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 ASYNCHRONOUS 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 ASYNCHRONOUS 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 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.

9.3.5 Pointer definition

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

10 CAM control blocks

The CCBs used by peripheral drivers and applications to request functions of the XPT and SIM have a common header, as shown in Table 2.

Table 2 – CAM control block header

Size	Dir	
4	O	Address of this CCB
2	O	CAM Control Block Length
1	O	Function Code
1	I	CAM Status
		Connect ID
1		Reserved
1	O	Path ID
1	O	Target ID
1	O	LUN
4	O	CAM Flags

The sequence of the fields in the data structures shall be consistent between vendors (i.e., the binary offset shall be the same for every field). The binary contents of fields may vary according to the memory addressing protocol of the processor.

The definition of the fields in the data structures can vary among 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.

A peripheral driver, the XPT, or a SIM that allocates CCB(s) shall be responsible for freeing those CCBs it has allocated after the allocating entity is finished with the CCB(s). A peripheral driver, the XPT, or a SIM shall not free any CCB that it has not allocated. A peripheral driver, the XPT, or a SIM shall not free a CCB that has been sent to `xpt_action()` and has not completed (see 0 for further information on CCB completion). A peripheral driver, the XPT, or a SIM shall not free a CCB more than once per allocation of that CCB and shall be responsible for keeping track of the CCB(s) it is using in a vendor unique manner.

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 3 summarizes the message support.

Table 3 – 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	

10.1 CCB header fields

10.1.1 Address of this CCB

Pointer containing the physical address of this CCB.

1.1.2 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.

10.1.3 XPT function code

The function codes used to identify the XPT service being requested are listed in Table 4.

Table 4 – XPT function codes

Code	
00-0F	Common functions
00h	NOP
01h	Execute SCSI I/O (see 11.2)
02h	Get Device Type
03h	Path Inquiry
04h	Release SIM Queue
05h	Set Asynchronous Callback
06h	Set Device Type
07h	Scan SCSI Bus
08-0F	reserved
10-1F	SCSI control functions
10h	Abort SCSI Command
11h	Reset SCSI Bus
12h	Reset SCSI Device
13h	Terminate I/O Process
14h	Scan Logical Unit
15-1F	reserved
20h	Engine Inquiry (see clause 12)
21h	Execute Engine
22-2F	reserved
30-3F	Target mode (see 11.4)
30h	Enable LUN
31h	Execute Target I/O
32h	Accept Target I/O
33h	Continue Target I/O
34h	Immediate Notify
35h	Notify Acknowledge
36-3F	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.

10.1.4 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 5.

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).